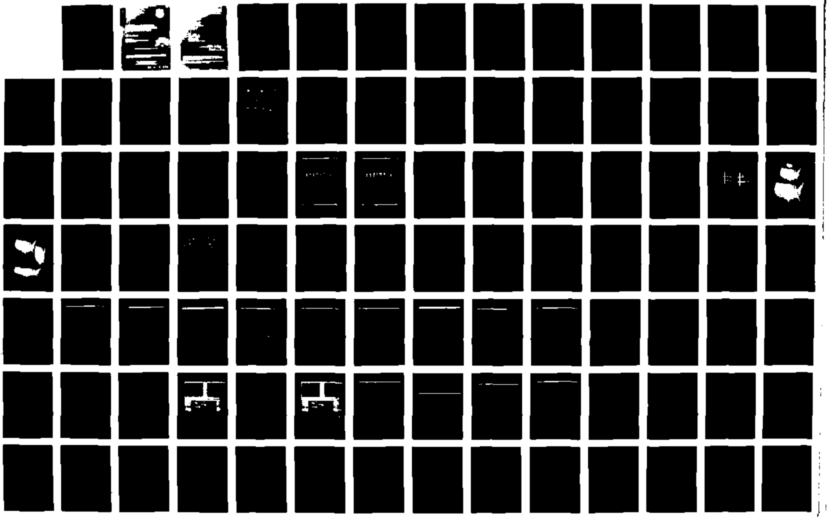
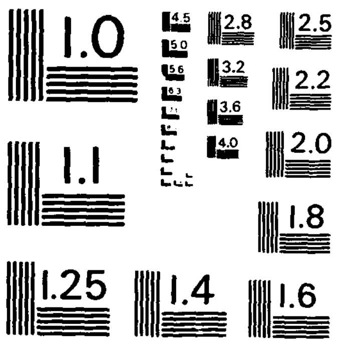
AD-A203 780 RESEARCH IN DISTRIBUTED PERSONAL COMPUTER-BASED INFORMATION SYSTEMS VOLUM..(U) BBN LABS INC CAMBRIDGE MA H C FORSDICK ET AL. AUG 88 BBN-5901-VOL-1 1/2

UNCLASSIFIED RADC-TR-88-159-VOL-1 F30601-81-C-0256 F/G 12/7 NL




1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

[illegible]

[illegible]

[illegible]

[illegible] agency or the U.S. Government.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

RESEARCH IN DISTRIBUTED PERSONAL COMPUTER-BASED INFORMATION SYSTEMS
Vol I (of two)

Harry C. Forsdick
Robert H. Thomas

Contractor: BBN Laboratories Inc.
Contract Number: F30602-81-C-0256
Program Code Number: XT10
Effective Date of Contract: 2 July 1981
Contract Expiration Date: 14 January 1985
Short Title of Work: Research in Distributed Personal Computer-Based Information Systems
Period of Work Covered: April 84 - October 84

Principal Investigator: Robert H. Thomas
Phone: (617) 873-3483

RADC Project Engineer: Thomas F. Lawrence
Phone: (315) 330-2158

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Thomas F. Lawrence, RADC (COTD), Griffiss AFB NY 13441-5700 under Contract F30602-81-C-0256.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A203 700

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 5901	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-159, Vol I (of two)
6a. NAME OF PERFORMING ORGANIZATION BBN Laboratories Inc. / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COTD)
6c. ADDRESS (City, State, and ZIP Code) 10 Moulton Street Cambridge MA 02238-0001	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency / 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-81-C-0256
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington VA 22304	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 62708E; PROJECT NO. D224; TASK NO. 01; WORK UNIT ACCESSION NO. 01

11. TITLE (Include Security Classification)
RESEARCH IN DISTRIBUTED PERSONAL COMPUTER-BASED INFORMATION SYSTEMS

12. PERSONAL AUTHOR(S)
Harry C. Forsdick, Robert H. Thomas

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Apr 84 TO Oct 84	August 1988	108

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Multi Media Message System
12	07		Distributed System
			Distributed Personal Computer Environment

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The primary focus of the personal computer task area is the development of an electronic message system called Diamond, which will run in a distributed personal computer environment. The message system will: support a user interface that exploits the capabilities of advanced single-user computers, handle messages that contain data other than text, have a distributed architecture, operate in a secure fashion, permit use from a variety of user access points, and have a transportable implementation. (KR)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas F. Lawrence	22b. TELEPHONE (Include Area Code) (315) 330-2158 / 22c. OFFICE SYMBOL RADC (COTD)

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This is the final technical report for Contract No. F30602-81-C-0256, entitled "Research in Distributed Personal Computer Based Information Systems." It summarizes the major accomplishments of the contract. In addition, it reports on work done between April 1984 and October 1984. Work done before April 1984 is described in contract semi-annual Reports 1 through 5, BBN Reports 4924, 5301, 5395, 5722 and 5723.

The report is organized as follows. The contract objectives and the major results of the contract effort are summarized in Section 2. Sections 3 through 9 report on contract activity during the final 6 months of the contract. Section 4 discusses work on the Diamond multimedia system and related activities. Work on Jade, the Jericho Pascal operating system, is described in Section 5. Section 6 presents our activities related to the development of the Jade programming environment. Work on the Interlisp system for the Jericho computer is described in Section 7. Section 8 discusses our work on Aleph. Section 9 describes recent Hermes maintenance activity.

2. SUMMARY OF CONTRACT ACTIVITY

Section 2.1 summarizes the objectives of the work done under the contract, and Section 2.2 summarizes our major accomplishments.

2.1 Project Overview

The tasks for this project fell into three broad areas:

1. Research in distributed personal computer systems;
2. Support for the Strategic C3 Experiment;
3. Maintenance of the Hermes electronic message system.

The project objectives in each of these areas are reviewed briefly below.

2.1.1 Distributed Personal Computer Systems

The primary focus of the personal computer task area was the development of an electronic message system, called Diamond, which was to run in a distributed personal computer environment. The Diamond message system was designed to:

- Support a user interface that exploits the capabilities of advanced single-user computers;
- Handle messages that contain data other than text (e.g., images, line drawings, speech);
- Have a distributed architecture;
- Operate in a secure fashion;
- Permit use from a variety of types of user access points;
- Have a transportable implementation.

The personal computers to be used in the initial implementation of the Diamond message system were Jericho computer systems. The portability of Diamond was to be demonstrated by moving the system to another comparable personal computer system.

It was expected that development of the message system would require work in a number of supporting areas, including:

1. Basic System Support.

Diamond was developed as an application program that executes on a collection of personal computers and shared resource computers interconnected by a high bandwidth local network. Diamond, as well as other applications, requires the support of "operating system" level software. The purpose of this software is to make the Jericho personal computer usable as a sophisticated, autonomous, single-user computer system. Development of the basic system support involved the design and implementation of storage management functions, bit map display functions, a multiple process capability, an interprocess communication facility, and support for the standard DoD network communication protocols.

2. Input/Output Support for a Variety of Data Types.

The Diamond message system was designed to handle messages composed of a number of types of information, including text, facsimile, graphics, and speech. This capability for multiple media communication required the development of software to support the input and output of these different types of data, and, in some cases (speech, sound and facsimile), system engineering to interface the personal computer systems with hardware required for the input/output of this data.

3. Distributed System Support.

Diamond was designed to execute on a distributed system architecture. Diamond and other applications developed for this environment require supporting software that enables personal computers to function effectively in a multiple-computer network environment. The supporting software required included: a network interprocess communication facility, a distributed file system supported by personal computer storage resources and dedicated file server computer resources, means for accessing devices that are remote from a personal computer as if they were local, a user authentication mechanism, and access control mechanisms to provide for controlled sharing in a distributed environment. The supporting software runs in part on the personal computers, and in part on the shared-resource computers.

4. Programming Language Support.

The Pascal programming language was used for much of the initial programming required for the Diamond message system development. In addition, it was expected that InterLisp would be used for experimentation and research in the user interface issues. Therefore, a certain amount of effort was required to ensure that the implementations of Pascal and InterLisp for the Jericho computer and their supporting environments are adequate. Furthermore, it was felt important that software modules written in Pascal and InterLisp be able to be used together in personal computer based systems such as Diamond. When the project began, this sort of interoperability was not possible, and it was not clear to what extent it could be achieved.

5. **Programming Environments**

Diamond was expected to be a reasonably large system. It would be built by a team of implementers, of which each member would use a personal comr .ter for software development. To facilitate implementation of systems to be built like Diamond, we planned to design, implement, and experiment with an application development environment, called the Jade environment. The Jade environment was intended to support the construction of distributed application programs and to be capable of supporting programming projects large enough to require many programmers, each supported by a personal computer.

We also planned to experiment with new programming environment, called Aleph, to explore extensions to the Interlisp environment that exploit features unique to personal computers of the Jericho class. This was to involve experimental investigation in the areas of graphical debugging, facilitation of routine bookkeeping activities, techniques for presenting multiple views of systems, vocal annotation of textual documents, and content-addressed documentation.

2.1.2 Support for Strategic C3 Experiment

The objective of work in this area was to support the Strategic C3 Experiment, a technology transfer and evaluation project conducted by DARPA and the Strategic Air Command. A number of contractors worked on this experiment with ARPA and SAC. Our role was to adapt the Hermes electronic message system to the needs of SAC users participating in the experiment.

In particular, we worked to:

1. Modify Hermes so that it can operate with a full-screen editor, such as EMACS or WE, in order to provide full-screen editing and composing of draft messages as an integrated Hermes functicn.
2. Extend the data management capabilities of Hermes to provide a template-driven report generator capable of summarizing the information contained in groups of message/records.
3. Investigate the problem of software aids for scheduling personnel and equipment. Develop algorithms and experimental software to support these scheduling tasks and experimentally study user interface and implementation issues.

2.1.3 Hermes Maintenance

The objective of this task was to provide software maintenance for the Hermes electronic message system. This included correcting problems that would prevent *effective* use of Hermes, should any arise, installing Hermes on new hosts at the direction of the ARPA office, and making improvements to the Hermes software.

2.2 Summary of Contract Accomplishments

Some of our accomplishments during the contract were the following:

- The Diamond multimedia message system was designed, implemented, and successfully demonstrated. Diamond currently allows text, graphics, images, speech, electronic spread sheets and charts to be combined into a single integrated document or message. Diamond is implemented as a distributed system which runs on an architecture that includes powerful single user workstations and shared server computers interconnected by a high performance network. Diamond has been in use by the project team for over a year, and will be available for use by others on commercially available hardware in the near future.

- The feasibility of transporting the Diamond system from a hardware base of Jericho workstations, upon which it was originally implemented, to other comparable hardware was demonstrated by transporting most of the Diamond implementation to a hardware base of Sun Workstations.

- An approach to permit programs written in Interlisp and in Pascal to run simultaneously on a Jericho computer was designed and enough of the design was implemented to demonstrate the feasibility of the approach. This approach is documented in BBN Report No. 5287, "Language Interoperability on Jericho."

- A Software State Database system was designed and implemented. This system facilitates software development in an environment where many machines are used to support many developers working on a project. The system maintains information on the software modules which make up the system under development. Information maintained about a module includes where (which machine) the most recent version of the file for the module resides, the person who last worked on the module, whether the module is currently being worked upon and by whom, other modules which depend upon the module, other modules upon which the module depends, and so forth. The system provides a number of facilities including convenient means for installing new versions of modules, checking modules out to work on, and distributing the latest versions of software to selected machines.

- A facility for monitoring the execution of distributed application programs was designed and implemented. The facility is designed to serve both as an aid for debugging distributed programs and as a means for demonstrating them. The facility makes it possible for a user to "watch" the execution of a distributed program at a number of levels of details.

- We participated with other DARPA contractors in the development of protocols that support transmission of multimedia documents among hosts in the DoD internetwork. These protocols make it possible for the users of a Diamond cluster to communicate over the internet with users of other Diamond clusters or of other multimedia message systems.

- The Jericho INTERLISP system was successfully ported from INTERLISP-10 and is in active use by the AI community at BBN. The system incorporates both reference counting and compacting garbage collection, bitmapped graphic in color and in black and white, and multiple-processing capability.

- We designed and implemented a Content-Addressed Documentation capability called "The INTERLISP Advertiser", and we provided the following programming tools: Directory and File Browsers, File and Code Comparison Presenters, Graphical Debugger, and Apropos and Indexing facilities.

- A number of modifications to the Hermes message system were made to support the Strategic C3 experiment. The principal modifications were to integrate the EMACS screen editor into Hermes in order to provide a full-screen editing capability for messages, and to extend the Hermes data management capabilities to support template-driven report generation. These modifications were integrated into the standard distribution version of Hermes, making them available to all Hermes users.

- The Hermes message system was maintained for the period of the contract. As part of this effort, Hermes was modified to operate in the DoD internet as part of the ARPANET transition for NCP to TCP; the Hermes distribution and installation procedures were simplified and documented so that systems personnel at Hermes sites could install new versions of the system will no direct involvement by BBN personnel; a number of minor enhancements were made to Hermes; and numerous minor problems and bugs were corrected.

3. OVERVIEW OF RECENT ACTIVITY

Sections 4 through 9 report work done during the last six months of the contract. This section summarizes project activity during that period.

Our major accomplishments during this period include the following.

- The Diamond multimedia document editor was successfully ported from Jericho to the Sun workstation early in the reporting period. Since we expected that the editor would be one of the more difficult parts of Diamond to port, this success gave us confidence that the porting strategy chosen was a good one.

- The Diamond multimedia editor was installed on the Sun workstations in the DARPA office. Draft documentation on the use of the editor was prepared and delivered to DARPA.

- After porting the multimedia editor, most of the other parts of Diamond have been successfully ported to the Sun workstation. The components ported include the interhost interprocess communication facility used by Diamond, the Diamond Access Point and all of the Access Point tools, the Document Manager, and the Authentication Manager.

- As part of an effort to understand and improve the performance of Diamond, we developed a number of performance monitoring tools which we used to conduct a series of performance measurement experiments on Diamond. After analyzing the results of the experiments, we were able to identify a number of performance bottlenecks, and eliminate many of them.

- The notion of workstation authentication was added to the Diamond Authentication Manager. Workstation authentication provides a means for a workstation to authenticate all of its processes in a single interaction with the Authentication Manager. The Diamond Access Point was modified to make use of workstation authentication. Use of workstation authentication results in a significant improvement in interactive responsiveness for Access Point operations that require the creation of a new Access Point process since there is no need to explicitly authenticate the new process by means of an interaction with the Authentication Manager.

- The Authentication Manager was improved by adding access control to the operations it performs.

- A means for permitting a number of Diamond users to serially share a single vocoding device was developed. A Vocoder Manager process, which manages the vocoder and an "intelligent" modem, was developed to support this. When a user at an Access Point needs to use a vocoder, the Access Point makes a request of the Vocoder Manager. If the vocoder is not in use, the Vocoder Manager uses the modem to place a telephone call to the user's office. After the call is established, the telephone circuit is used as the path between the user and the vocoder.

- The EditDoc multimedia editor has been improved in a number of ways. The notion of a clipboard has been implemented; a user can use the clipboard to "cut" parts of a document and to "paste" them into other documents (or into another part the same document). Support for electronic spread sheets as a media type has been added; spread sheet data can be represented in tabular form or in a variety of chart forms or in a combination of both tabular and chart forms. The text handling capabilities of EditDoc have been improved to support dynamic formatting of text as it is entered and edited; in addition, the options for formatting text have been greatly expanded. The image editing and graphics editing capabilities of the editor have also been enhanced.

- o The Software State Database System, which is part of the Jade environment, has been improved in a number of ways. These improvements include: the ability to recognize dependencies among files in the data base; the use of dependency information to check the consistency of operations (such as the installation of modified modules); support for moving modules back and forth between private software development environments and the public software repository; and, the maintenance of creation (write) dates for each module being maintained by the system. In addition, the system has been made significantly more robust.

- o The initial implementation of the IPC Monitoring Facility, which is a part of the Jade environment, has been completed, and it has been released for general use within the project.

- o We implemented a version of Active Values, which provide a way to invoke a function when the value of a datum is set or accessed. This opens the way for the incorporation of Object Oriented programming paradigms in Jericho INTERLISP.

- o We designed and implemented an application of Active Values that allows users to monitor and visualize the changes that occur in any given datum as a consequence of program activity or editing. We also improved the Code Presenter and the Graphical Debugger.

- o We attended the IFIP 6.5 Working Conference on Computer Message Systems held in May 1984 at Nottingham England and presented a paper on Diamond titled "Initial Experience with Multimedia Documents in Diamond".

These items, and others, are described in the following sections.

4. THE DIAMOND MULTIMEDIA MESSAGE SYSTEM

Diamond is a distributed system implemented by a variety of components which together provide a single coherent service. The components of Diamond are:

- **User Access Point**: The user's main contact with Diamond. The Access Point is composed of several tools including:
  - **Coordinator**: All of the actions of the Access Point are directed by this tool. The user can always inquire about the state of Diamond by interacting with the Coordinator.
  - **Document Presenter/Editor**: Documents are viewed and composed using this tool. The Document Editor embodies all of the protocols concerning Document and Atomic Object Representations.
  - **Folder Presenter**: Folders of documents and other folders are viewed and manipulated by the Folder Presenter. This tool also interacts with the Document Presenter/Editor tool to show or compose documents.
  - **User/Group Registry Presenter**: The User and Group Databases (see below) can be examined and modified using this tool.
- **Authentication Manager**: This component maintains information about authenticated users and processes of a Diamond cluster as well as long term information about user preferences and groups of users.
- **Document Manager**: Documents and folders of documents and other folders are managed by this component. When a user saves a document, the Document Manager accepts the document and stores it in a Folder for later retrieval.
- **Device Managers**: Various devices such as Image Scanners and Printers are managed by Device Managers.
- **Import/Export Manager**: Documents sent to recipients outside a Diamond cluster are Exported by this component. Likewise, documents originating outside a cluster which are addressed to a recipient supported by the cluster are imported by the Import/Export Manager. This component takes care of any protocol conversions that may have to occur between the standard DARPA Internet Multimedia Protocol and the protocols used internally by Diamond.
- **Internet Gateway**: Communication with hosts on the DARPA Internet is done by use of the Internet Gateway.

Figure 1 illustrates the architecture of a Diamond cluster.

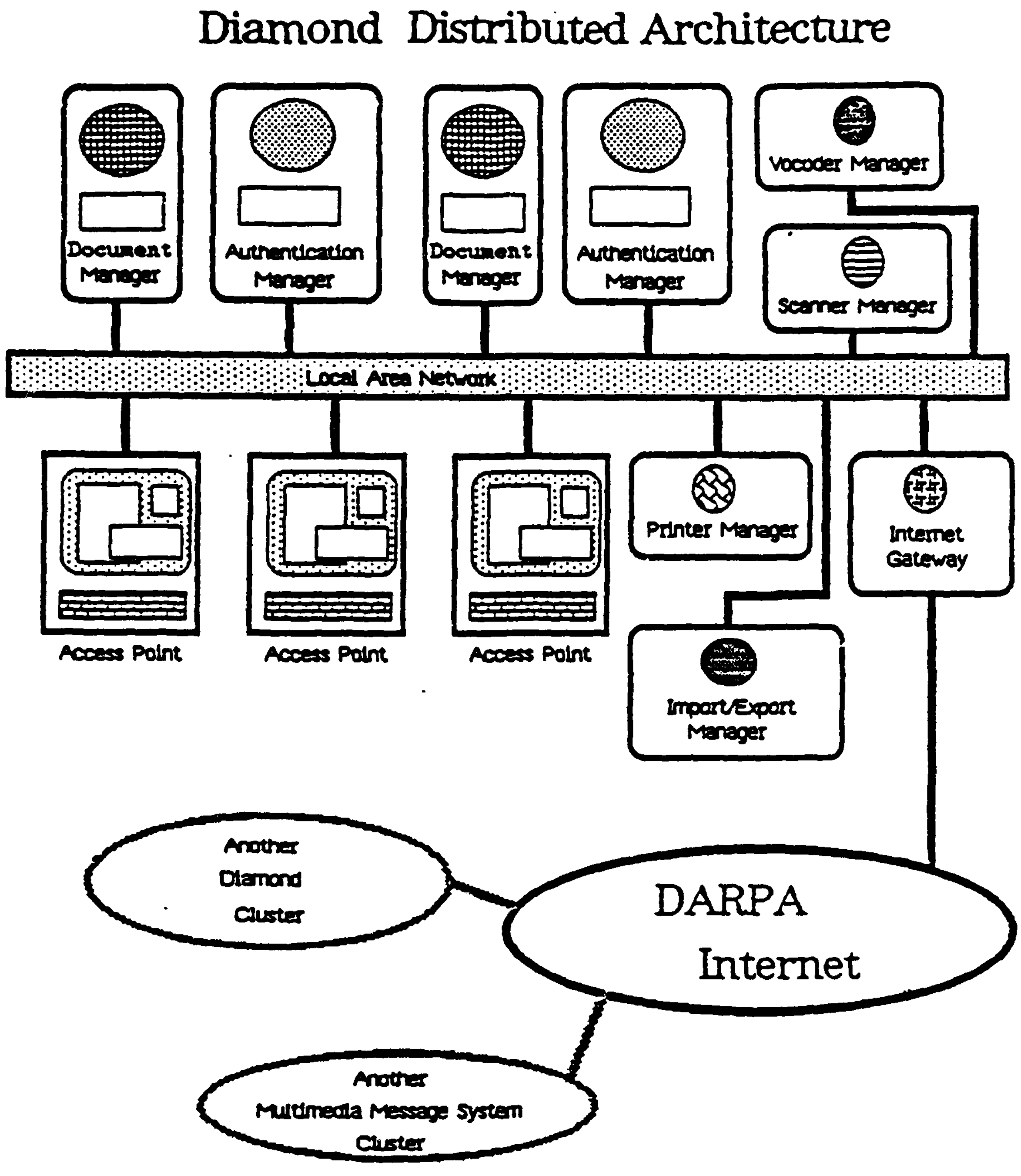


Figure 1. The Distributed Architecture of Diamond

4.1 Porting Diamond to the Sun Workstation

We have successfully ported the EditDoc multimedia editor from Jericho to the Sun workstation. EditDoc running on a Sun was demonstrated at BBN to DARPA in mid April. The version that runs on the Sun is identical to the version that runs on Jericho. This version of EditDoc has been installed on the Sun Workstations in the DARPA/IPTO office. Preliminary documentation on the use of EditDoc has been prepared and delivered to DARPA.

Work has progressed on porting the rest of Diamond to the Sun workstation. The following sections describe that work.

4.1.1 PCode-to-68000 Code Peephole Optimizer

The major work done on the PCode translator during this reporting period beyond the correction of bugs as they were discovered was the addition of a peephole code optimizer. The optimizer makes a pass over the generated code looking for code sequences which can be transformed into better code sequences. Ordinarily, a peephole optimizer functions by limiting the context within which it operates to a small number of lines of output code. This optimizer, however, operates on the entire body of a procedure.

The first step in making this change was to modify the code generator output format from strings to a more easily processed form. Then the generated code was examined manually looking for bad code sequences, and transformations were devised to improve them. The transformations fall into a relatively small number of categories:

1. Choosing the best way to load constants into registers and variables.
2. Adding, subtracting, and comparing small constants.
3. Condensing chains of logical operations.
4. Eliminating unreachable code.
5. Testing booleans in packed records.
6. Using to memory operations where possible.
7. Eliminating eliminating superfluous TST and CMP instructions.
8. Using word and byte instructions were possible.

9. Using short form branches were possible.

A total of 27 transformation have been implemented which result in a code size reduction of about seven percent. Code speed improvement has not been measured, but several of the transformations produce substantial improvement in the loop increment and testing code.

4.1.2 Database Server

One of the biggest differences between the programming environments of the Sun and the Jericho is the per-process address space on the Sun versus the single address space for all processes on the Jericho. Memory sharing on the Jericho allows very wide-band communication to be implemented efficiently. Several Diamond servers (the Document Manager and the Authentication Manager) made use of this facility to communicate with sub-processes which were spawned to handle single requests. In particular, these servers made use of several shared, long-term, persistent databases. Access to the databases was synchronized by the use of semaphores, but changes were propagated automatically since there was only one copy of the database in memory, directly reflecting the copy of the database on disk.

In order to allow multiple processes to share the same database on the Sun, we developed a single database server which maintains a list of currently open database files. It handles database requests (Find, Insert, Delete) from processes on the same machine. The procedural interface for contacting the database server is exactly the same as the interface for directly dealing with the database file. This has minimized the changes necessary to programs using the database facility. The overhead in accessing a database entry managed by the server (Jade IPC, copying of data) is obviously larger than using shared memory but is acceptable, given the way the database is used. A typical Document Manager or Authentication Manager operation will access or update only a few entries. Inter-machine communication costs far outweigh the additional intra-machine communication imposed by the database server approach.

The database server is being used on both the Jericho and the Sun.

4.1.3 First Release of Diamond on Sun Workstation

By the end of the reporting period we had ported the core of Diamond to the

Sun Workstation and have demonstrated that the basic operations of Diamond can now be performed on the Sun Workstation.

The components that have been ported to the Sun to date are:

- o Access Point and Access Point tools:

  EditDoc
  Coordinator
  ShowFolder
  ShowRegistry.

- o Document Manager

- o Authentication Manager

Components remaining to be ported are:

- o Import/Export Manager

- o Scanner Manager

- o Voice Manager

In addition, a Printer Manager will have to be written once a "standard" Laser Printer (or set of printers) has been chosen. Since one of the first Diamond installations is expected to be in the ADDCOMPE testbed, it is likely that (one of) the standard printer(s) will be the printer chosen for use in ADDCOMPE.

In the next section we describe some performance improvements that have been made to the version of Diamond running on the Jericho computer system. The current version of Diamond on the Sun has some additional performance problems which will have to be solved in the future.

4.2 Performance Analysis of Diamond

During this reporting period, a substantial amount of work has been done on measuring, analyzing, and improving the performance of various parts of the Diamond system. The work falls into three broad categories. First, a collection of performance monitoring and analysis tools was developed, and various parts of Diamond were instrumented to use these tools. Second, a series of controlled experiments were performed to gather data, and analysis was done on those data. Finally, various performance improvements were designed, implemented, and validated with more performance measurement and analysis.

4.2.1 Performance Analysis Tools

There are three basic approaches to performance analysis that we have pursued. First, an event logging facility was implemented, various parts of Diamond were instrumented to use it, and an event log analysis program was implemented. This approach provides insights into the macro-behavior of the system. Second, the Jade *RoutineTrace* program was augmented and used to analyze micro-behavior of various parts of Diamond. Third, a model was built to describe the expected message passing behavior of the system for various tasks. The model used in conjunction with the log analyses showed us where inefficient exchanges of messages were taking place. The model, the logs, and the routine traces were also used to validate each other, to insure that we were not overlooking anything.

The event logging mechanism is made up of two parts, a library called *LogLib* which is used by Diamond components as the interface to the logging mechanism, and a server called *LogServer* which runs in a separate process and serializes the events and records them in log files. An event contains the following information:

- A timestamp;
- The time spent in pre-processing (time is recorded for elapsed wall clock time, CPU utilization time, and page fault time in milliseconds);
- The time spent in the body (i.e., the time required to perform the requested task);
- The time spent in post-processing;
- The operation being performed;
- The UID of the object being manipulated;
- The source of the event.

The event log analysis program is called *PrintLog* and is used in two primary ways. First, it is used to produce summary statistics to aid in identifying the most frequently used operations and the most costly operations. Second, it is used to examine sequences of events, to fully understand the message passing behavior and where time is being spent. This is particularly difficult in a distributed environment. The analysis tool can integrate logs from different machines to make the total sequence of events more understandable.

4.2.2 Experiments and Results

In order to fully understand the performance of the system, data was collected under several different circumstances. First, data gathering was enabled on all of our machines, and a large mass of field data was collected over several weeks. This data was used mainly to identify the most frequent and the most costly operations. Second, a controlled experiment was done between two machines, with nothing else running on either machine. This data was used to identify common sequences of messages that should be implemented as compound operations, and extraneous messages that could be eliminated. After compound operations were implemented and extraneous messages were eliminated, the experiment was repeated. It became clear from this data that paging performance was the major bottleneck. A series of routine traces was done in various parts of the system to identify paging problems. Some of those problems were corrected, others were not easy to correct, yet tended to mask out other performance problems. So, the third series of experiments involved adding memory to two machines and repeating the second experiment, effectively eliminating the paging problems by brute force. This final experiment provided more detailed and reliable information about where time was being spent in the system.

The most significant performance improvements were the identification of common sequences of operations that could be turned into compound operations. Most of these changes were in the Document Store Manager, the Access Point Coordinator, and the Folder Presenter.

The Document Store Manager was improved in several ways. It was modified to make more efficient use of the underlying database system that it uses to keep track of documents and parts of documents. It was modified to minimize the number of times it opens and closes files in the process of handling user requests to create and retrieve documents. The implementation of the *Send* operation, which sends a document to other users as a message by adding citations for the document to their InBoxes, was streamlined and made less synchronous in nature.

The Access Point Coordinator was changed to initialize processes and create windows in a much faster way. Also, the login procedure and user preference mechanism was significantly streamlined. The Access Point now uses "workstation authentication", a mechanism that was added to the Authentication Manager to allow a single authentication for a workstation, with possible overriding process authentication (see Section 4.3). This cuts down considerability on message traffic when a new process is spawned by the Coordinator.

The Folder Presenter was changed to deal with citation display formats in a much more efficient way. This makes a substantial difference for large folders. Also, when a user asks to *Open* a set of documents, the Folder Presenter initiates a parallel pre-fetch of those documents. This makes a big difference when the user uses the *Next Document* command in EditDoc, since the next document will already be in the local cache.

The results of all of these measurements and improvements has been significant. Before we started this process, the system was useable for everyday message traffic, but was very frustrating. It is now very nice to use, and usually seems as fast of most text-only message systems. However, the version of Diamond that is now running on the Sun Workstation still has serious performance problems. A similar round of performance measurement, analysis, and improvements will need to be made on that version of the system.

4.3 Authentication Manager

Two major improvements have been made to the Authentication Manager: the notion of workstation authentication has been implemented, and access control has been added to Authentication Manager operations. In addition, a number of minor extensions were implemented, and the Authentication Manager was transported to the Sun Workstation. Finally, a review of the Diamond access control mechanism, motivated by difficulties experienced by users in setting up the access controls they desire, was started.

4.3.1 Workstation Authentication

Many different processes run on behalf of a user interacting with Diamond on an Access Point workstation. For example, for each open folder, there is a process on the Access Point running the ShowFolder program. Under the initial design for the Authentication Manager, each of these processes has to have an entry in the Access Control Database (ACDB) identifying the principal for which it has been authenticated. The management of this information turns out to be quite time consuming and was identified in the performance analysis task as a potential area for improvement.

To reduce the time spent in managing this information, the concept of an authenticated workstation was developed. Under this scheme, a host (typically a single user workstation) can request that every one of its processes be bound to the same principal (user). That is, all of the processes running on a given workstation can be authenticated with one entry in the ACDB. The workstation is identified by a process UID with the Internet Address of the workstation and 0's for the Incarnation and Sequence numbers. When a user enters Diamond he identifies the principal for which all processes running on the workstation should authenticated. This authentication is kept in the ACDB until the user logs out. To permit finer grain authentication for multiple processes running on the same workstation which need to be associated with different principals, when the *AuthorizationBindingOf* operation is invoked, the ACDB is first searched for the Unique Identifier of the specified process. If there is no match, then the database is searched for the UID of the workstation on which that process resides (gotten by setting the Incarnation and Sequence numbers of the process's UID to 0).

Use of workstation authentication can result in a substantial performance improvement in many situations. For a system like Diamond where the access point/user interface is supported by a dynamically changing collection of processes, the number of interactions between the user interface and the Authentication Manager is significantly reduced. The workstation need interact with the Authentication Manager only once to authenticate itself and all processes that (will) run on it, instead of interacting with the Authentication Manager each time a new process is created to handle some user request.

4.3.2 Access Control Within the Authentication Manager

The initial version of the authentication manager did not perform any access checks on operations performed on the registry or its contents. This omission was made to facilitate the initial implementation and is clearly not permissible for an operational system. During this reporting period, access control checking was added to limit the operations which may be performed by any particular individual.

Each record describing a principal or group in the registry has an associated access control list. This access control list has the same form as access control lists for other kinds of objects, such as documents; it lists the access rights for each of several principals or groups. This access control list specifies who may perform each of several different operations on that record. Examples of operations are reading and writing user parameters, listing the groups of which a principal is a member, listing the principals and groups which are members of a group, and reading and writing the access control list itself.

In addition to the access control list per database record, there are default access control lists used to establish the access control lists of newly created principals and groups and an access control list for the registry itself. The registry access control list determines who may perform operations such as creating and removing principals and groups. The default access control lists are implemented as a special principal and a special group and may be edited using the operations available for editing ordinary access control lists. The registry access control list is built in to the authentication manager and is not editable. It includes only the SystemAdministration group.

4.3.3 Authentication Manager Extensions

Certain operations in the authentication manager which had been planned but not yet implemented were completed in this reporting period. These include the operations to read and modify access control lists and the operations to add or remove groups from the enabled set (a mechanism to reduce the access a user may exercise).

A number of other improvements/changes were also made. The format of database records was changed both to add access control lists and to reduce their size by reducing the size of the area used to hold names and passwords. The hash table routines containing the authentication database (record of logged in users) were recoded to reduce memory requirements. Old entries in the authentication database are removed when newer entries are made. Previously, entries would accumulate for as long as the authentication manager was operating.

4.3.4 Port to the Sun Workstation

The authentication manager has been ported to the Sun workstation. This task primarily involved changes to circumvent the lack of memory-sharing facilities on the Sun. The three databases (Principal Registry, Group Registry, and Authentication Database) are manipulated both by the main program and by sub-processes created to handle individual operations. The two registries are standard databases and only required modifications to use the new database server described in Section 4.1.2. The authentication database manipulations were altered so that the main process actually maintains the database and the subprocesses send interprocess messages to make modifications.

4.3.5 Review of Diamond Access Control

Diamond uses an access control list mechanism for access control. Each object, such as a multimedia document, has associated with it an access control list which specifies who can access it and in what what ways. Each entry on an access control list includes a principal (user) or group id and the type of access the principal or group is permitted. Every object in Diamond has a type which determines the operations that can be performed upon it. Originally, the type of access permitted to a principal or group was specified by listing the operations permitted on the object. In order to create a new access control list entry or modify an existing one, the user specified allowable access in terms of permitted operations. This approach has the difficulty that users tend to think in terms of relatively high level operations, which are usually implemented by several low level operations on objects, and the operations that appear on access control list entries are the low level operations. Consequently, although in principle the access control mechanism is quite simple and easy to understand, in practice users tend to have difficulty understanding how to specify access restrictions to achieve the access control results they desire.

To correct this situation we have initiated a review of the Diamond access control design, and expect to modify it slightly to enable users to specify access in terms of more "macro" operations. These macro operations will be mapped into the low level operations actually appearing on the access control list entries.

4.4 Document Manager

During this period, work on the Document Manager focused on performance improvements, operational improvements, bug fixing, and porting the Document Manager to the Sun workstation.

The Diamond Document Manager users Jade files to store Diamond folders, documents, and components of documents. With the Jericho Jade file system the time required to open a file in a given directory increases with the number of files catalogued in the directory. During this reporting period we changed the way in which Jade files and directories are used to implement the Document Store. The new strategy is designed to spread the files used to store Diamond objects across more directories in order to minimize the time required to open the files necessary to access the objects. As new Diamond objects are created, whenever the number of files in a given directory reaches a pre-determined threshold, a new directory is created and subsequent files for new objects are created in the new directory.

We encountered a problem with the Document Store Scavenger program[1]. The problem was discovered when the Scavenger was used to reconstruct the Document Store after a system crash. The Scavenger correctly rebuilt the document hierarchy, but failed to restore the access control lists for the various documents, folders and media objects. The source of the problem was that access control lists for Diamond objects were not stored with the objects themselves (i.e., in the files used to implement the objects), but rather were stored with object descriptors (i.e., in the internal tables used to keep track of the objects). This was corrected by modifying the Document Manager to store access control lists with the objects themselves.

We have developed a tool that can be used to remove folders and documents from the local Document Store cache[2]. The tool, which is called TrimCache, "trims" the cache so that the storage consumed by the cache is less than some specified amount of disk storage. It does this by deleting cached objects, least recently used objects first, until the desired storage level is reached. TrimCache is typically run periodically on Access Point workstations in the background via the Background Server system.

As part of the effort to port the Document Manager to the Sun Workstation the Diamond Document Manager was modified to make use of the Database Server (see Section 4.1).

[1]The Scavenger is a program that reconstructs the Document Store by scanning all of the files used to store documents and folders. It is useful if the tables internal to the Document Store are damaged due to a system crash or a software bug. See Semi-Annual Technical Report No. 5, BBN Report No. 5723 for more detailed information about the Scavenger.

[2]In order to improve performance, each Access Point workstation maintains a local cache of recently referenced folders, documents, and atomic objects. The Document Store cache is described in more detail in Semi-Annual Technical Report no. 5, BBN Report No. 5723.

4.5 Vocoder Manager

At present Diamond uses the LPC vocoding devices designed by Lincoln Laboratory for voice i/o. The Lincoln vocoders are currently available only as prototypes, and, consequently, are relatively scarce. In order to make voice more widely available as a Diamond media, we have developed means for a collection of Access Point workstations to (serially) share a single vocoder.

A Vocoder Manager process is responsible for managing the shared vocoder. It runs on a server host to which the LPC vocoder to be shared has been interfaced. The Vocoder Manager host is also equipped with an "intelligent" modem with auto dialing capability.

When an Access Point needs to use the vocoder, it interacts with the Vocoder Manager to request use of the vocoder. For example, to output a voice passage in a message being presented to its user, the Access Point sends the voice passage and the telephone number for the user's telephone along with a "playback" request to the Vocoder Manager. The Vocoder Manager then places a telephone call to the user via the autodialer. After the call has been established, a path exists from the Vocoder Manager through the vocoder, the modem and the telephone network to the user. The Vocoder Manager can then output the voice passage through this path to the user. When a vocoder is needed to input or edit a voice passage, the Access Point initiates a similar sort of interaction with the Vocoder Manager; after the user is connected to the vocoder through the telephone network the input or editing can occur. While editing, the user sees a waveform and a "bouncing ball" which follows the visual waveform as he hears the voice. This is accomplished by a UDP packet stream between the Voice Server and the Access Point, letting the editor know where the vocoder is in the passage.

When low cost vocoders become widely available, dedicating one to each multimedia workstation will be preferable to using them in this fashion. However, we've found that sharing a vocoder through the Vocoder Manager works quite well. There are probably several reasons for this. Voice i/o is required relatively infrequently; this permits a single vocoder device to be shared among many users with little contention for it. The telephone system is pervasive, and its use is universally well understood; hence using it for voice i/o is natural, and requires little user training.

The Diamond Vocoder Manager demonstrates the feasibility of providing a voice i/o capability for Access Points not equipped with vocoders. With suitable adaptation, it would be capable of supporting voice i/o for low cost multimedia Access Points.

4.6 Access Point

A number of changes have been made to the Access Point during this reporting period. They fall into three categories: performance enhancements, the addition of error recovery mechanisms, and changes made to port the Access Point to the Sun Workstation. In addition, the Diamond Access Point has been modified to take advantage of workstation authentication to support workstation authentication (see Section 4.3). The performance enhancements have already been described in Section 4.2.

The error recovery mechanism, which had been designed earlier, has been fully implemented in the Access Point. Three classes of errors are handled. First, there are a set of errors indicating that the user is no longer properly authenticated. This is usually caused by the Authentication Manager being taken down and coming back up with no one being authenticated. This class of errors is handled by doing an automatic reauthentication; a window is popped up telling the user what is happening, then the reauthentication procedure ensures that the workstation is properly authenticated (which may require asking the user to retype his password). After the reauthentication procedure, the failed operation is retried. The second class of errors are communication failures. The response to these errors is to inform the user and ask if he wants to retry the operation or quit. The third class of errors is made up of all other errors. These errors are simply reported to the user. If such an error occurs in a multi-argument operation, the Access Point will reread the object to ensure that it has a faithful copy of it.

The initial port of the Access Point to the Sun Workstation has been completed. The most significant changes required were forced by three factors:

1. The Access Point Coordinator made much use of shared memory. Jericho supports memory sharing among processes, but the Sun does not.
2. The Coordinator manipulated windows that it did not own. The Jericho window system permits this, but the Sun window system does not.
3. The Coordinator and the Access Point tools were written assuming that the window system "repairs" any damage done to windows as they are uncovered

after being partially or completely buried by other windows. The Jericho window system repairs window damage for application programs, whereas the Sun window system does not. Instead, it delivers "window damage" hints to the application, signalling it that its window has been damaged. Window damage hints can arrive at any time. Hence the structure of the user interface implementation had to change to allow propagation of those hints back to the higher levels, where the display could be reconstructed.

If Pascal had either an exception handling mechanism or the ability to pass procedures as arguments, the problems posed by the third factor would have been much easier to solve.

The changes to the Access Point that were required to make it run on the Sun are typical of those that would be required to port any Jericho application that makes extensive use shared memory and of windows and graphical interaction. Because we feel the problems we encountered in porting the Access Point are generic in nature, we describe them in Section 5.2 which is concerned the issue of Sun / Jericho compatibility.

4.7 Multimedia Editor: EditDoc

Semi-annual technical report No. 5 reported on the design and initial implementation of Editdoc, the Diamond multimedia editor. A number of improvements made to EditDoc during this reporting period are described in this section.

4.7.1 Image Editing

The image editing capabilities of EditDoc have been significantly enhanced by the addition of the ability to scale and rotate images by arbitrary amounts. Scaling is specified by "dragging" a corner of the image with the mouse. Rotation is specified by placing a line segment on the image and moving one of its end points around the other end point. Both the rotation and the scaling operations are quite fast and are nearly able to follow the mouse in real time. The technique used to rotate and scale images are described in Section 5.1.

The image editing capabilities have been extended by adding the ability to edit a part of an image. Previously most image editing operations (e.g., scale, rotate, reflect) had worked only on the entire image. Finally, the ability to "paint" and erase images with a variety of brush shapes and textures has been added.

4.7.2 Spread sheets / Charts

A new object type, electronic spread sheets, has been added to Diamond. This involved integrating support for the composition, manipulation and presentation of spread sheets into EditDoc, and augmenting the representation of multimedia documents to include the spread sheet data type.

Support for spread sheets includes support for graph and chart representation of spread sheet data, as well as for tabular representation of it. That is, we have combined the ability to compute models using a spread sheet with the ability to automatically chart values in the spread sheet. There are several alternative ways to chart data including bar, line, point and pie charts.

A spread sheet / chart occupies a rectangular area of data. The rectangular area may be split up into separate panes in which separate areas of the spread sheet or a chart based on values in the spread sheet may be viewed. In its viewing form, a spread sheet / chart is just a table of values, possibly with ruling lines to give emphasis to groupings of data and/or a chart. Figure 2 shows a spread sheet in the form that a reader of a message would see it. In its editing form, a spread sheet / chart looks like a standard spread sheet with labels for rows and columns and/or a standard business chart. Figure 3 show the same spread sheet / chart in the form that the author of a document would see it. Charts have the same appearance in viewing and editing forms.

A rich set of functions is available in the spread sheet covering the following areas:

o Arithmetic Functions:

    Sum(of list), Average(of list), Min, Max,
    Round, Ceil, Trunc.

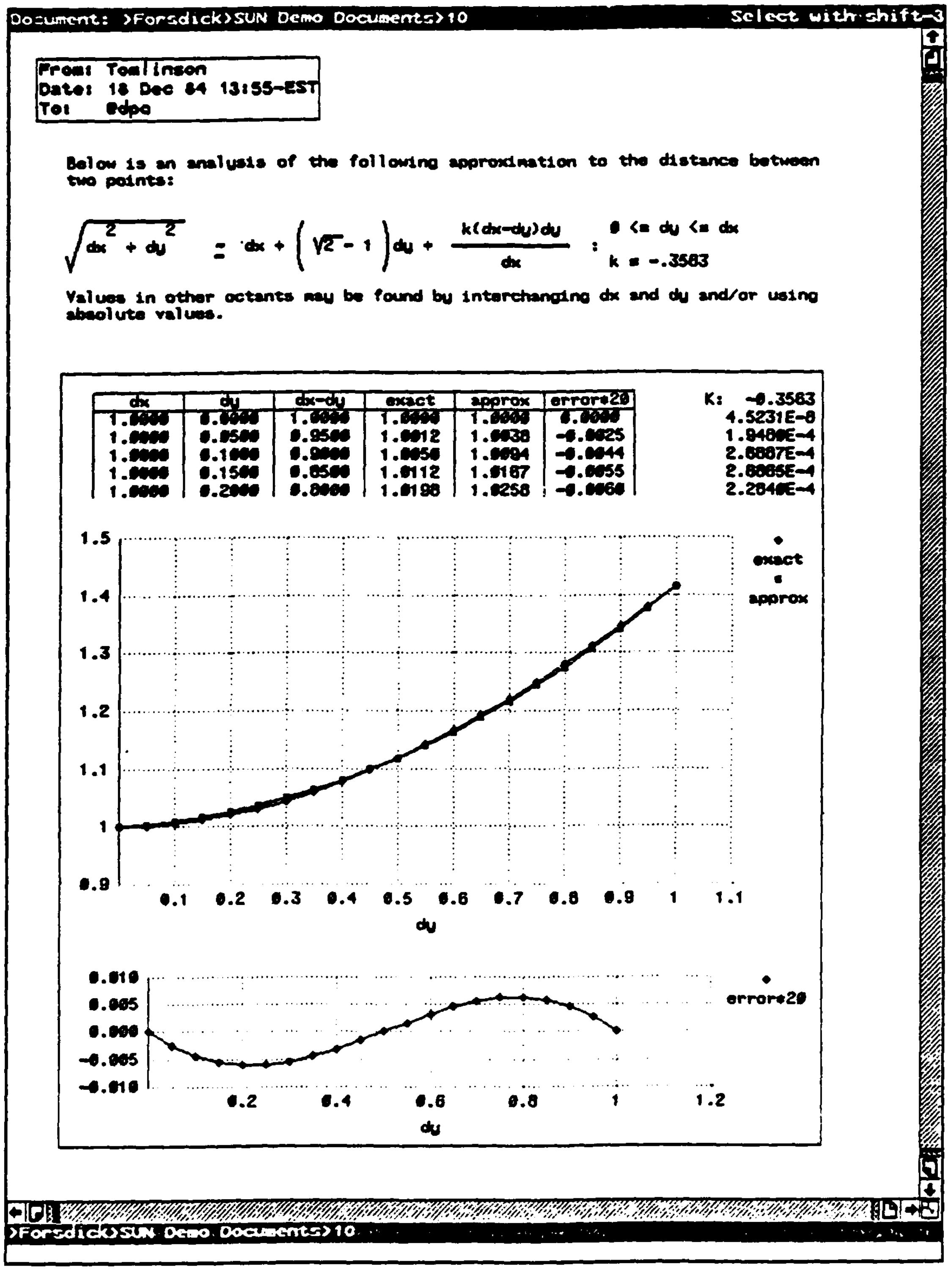
o Transcendental Functions:

    Ln, Exp, Log, PWRofTEN.

o Trigonometric Functions:

    Sin, Cos, ATan2.

o Logicel Functions:



Document: >Forsdick>SUN Demo Documents>10 Select with shift-3

From: Tomlinson
Date: 18 Dec 84 13:55-EST
To: @dpo

Below is an analysis of the following approximation to the distance between two points:

$$\sqrt{dx^2 + dy^2} \approx dx + \left(\sqrt{2} - 1\right) dy + \frac{k(dx - dy)dy}{dx} \quad : \quad \begin{matrix} 0 <= dy <= dx \\ k = -.3583 \end{matrix}$$

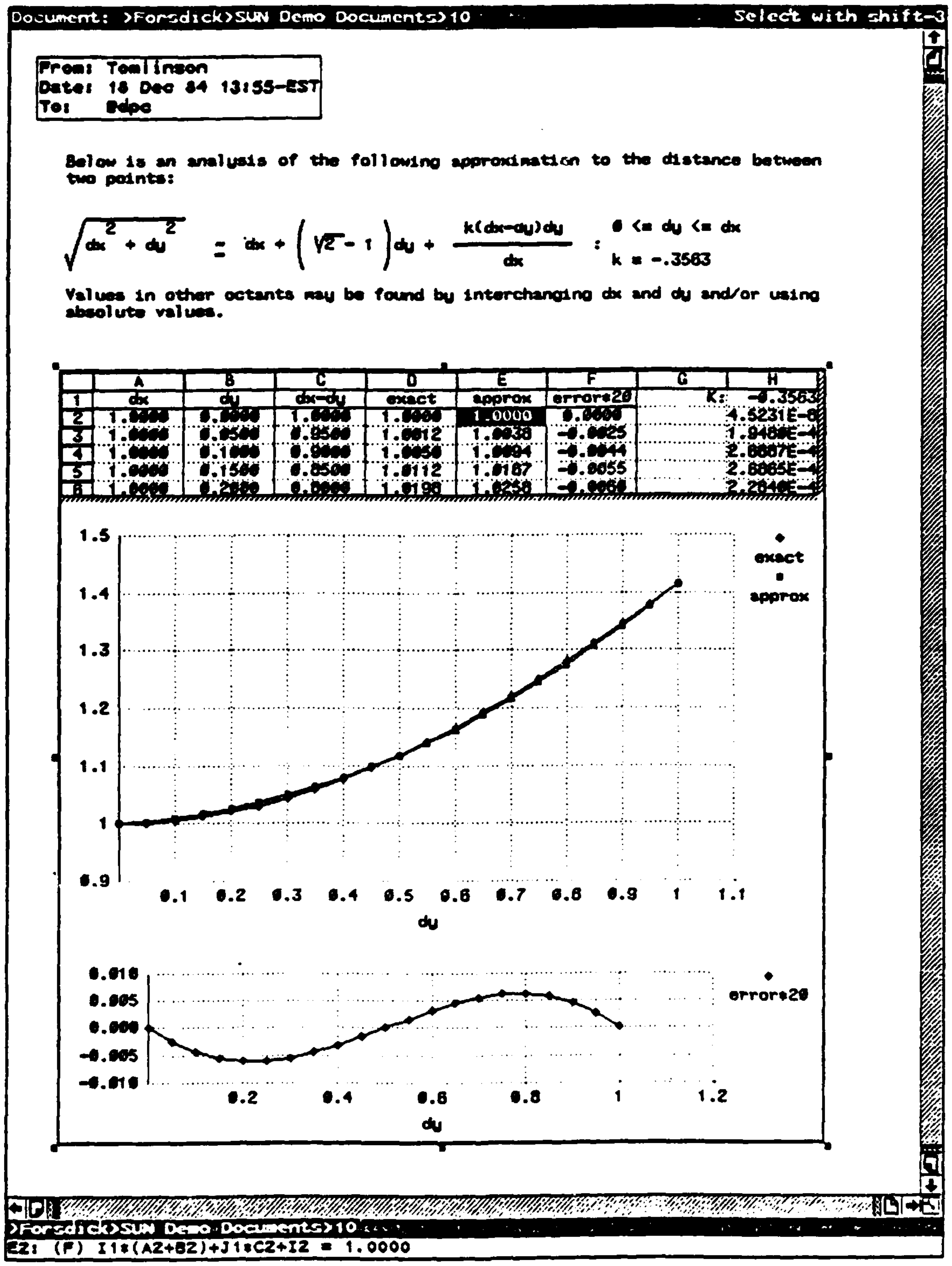
Values in other octants may be found by interchanging dx and dy and/or using absolute values.

dx	dy	dx-dy	exact	approx	error*20
1.0000	0.0000	1.0000	1.0000	1.0000	0.0000
1.0000	0.0500	0.9500	1.0012	1.0038	-0.0025
1.0000	0.1000	0.9000	1.0050	1.0094	-0.0044
1.0000	0.1500	0.8500	1.0112	1.0167	-0.0055
1.0000	0.2000	0.8000	1.0198	1.0258	-0.0060

K: -0.3583
4.5231E-8
1.9480E-4
2.8887E-4
2.8865E-4
2.2840E-4

>Forsdick>SUN Demo Documents>10

Figure 2. A Spread sheet / Chart in Viewing Form


Document: >Forsdick>SUN Demo Documents>10 Select with shift-3

From:	Tomlinson
Date:	18 Dec 84 13:55-EST
To:	@dpc

Below is an analysis of the following approximation to the distance between two points:

$$\sqrt{dx^2 + dy^2} \simeq dx + \left(\sqrt{2} - 1\right)dy + \frac{k(dx-dy)dy}{dx} \quad : \quad \begin{array}{l} 0 <= dy <= dx \\ k = -.3583 \end{array}$$

Values in other octants may be found by interchanging dx and dy and/or using absolute values.

	A	B	C	D	E	F	G	H
1	dx	dy	dx-dy	exact	approx	error*20	K:	-0.3583
2	1.0000	0.0000	1.0000	1.0000	1.0000	0.0000		4.5231E-8
3	1.0000	0.0500	0.9500	1.0012	1.0038	-0.0025		1.9480E-4
4	1.0000	0.1000	0.9000	1.0050	1.0094	-0.0044		2.8007E-4
5	1.0000	0.1500	0.8500	1.0112	1.0187	-0.0055		2.8065E-4
6	1.0000	0.2000	0.8000	1.0198	1.0258	-0.0060		2.2040E-4

>Forsdick>SUN Demo Documents>10

E2: (F) I1*(A2+B2)+J1*C2+I2 = 1.0000

Figure 3. A Spread sheet / Chart in Editing Form

And, Or, Not, If.

- o Finance Functions:

  Compound, Annuity, NPV.

- o Date Functions:

  Sum and Difference.

4.7.3 Text

A new sub-editor for text has been implemented. The goals for this new sub-editor were twofold: to dynamically format text as it is being entered and edited, so that the format is always accurately displayed in the EditDoc window; and, to greatly expand the formatting options available.

The new text sub-editor supports the following features:

- o There is one view of text. Text is dynamically formatted as it is entered from the keyboard. Justification and filling are done as keys are typed.
- o Formatting environments are supported so that an author can compose fully formatted documents. There is a set of default environments (Paragraph, Verbatim, Enumeration, Itemization) that can be used as the basis for additional user defined environments.
- o Document structure is preserved so that partially completed documents and cooperatively developed documents can be composed.
- o Font specification is by three parameters: Family, Style and Size.

  As a first step toward developing a collection of fonts, we have modified our font editing tool, EditFont, to accept a font and to produce italicized and bold versions of the font. The result of italicizing or "boldifying" a font is generally a reasonable approximation to the italic or bold version of the font; the font editor can then be used to "fine tune" individual characters in the font.
- o Editing actions may be performed by using the mouse to point to the text and make menu selections, and/or by means of keystroke commands which are similar to the EMacs command set.

We have plans for improving the way in which text objects are included in multimedia documents and will pursue these plans in future work.

4.7.4 Graphics

The Graphics handling capabilities of EditDoc have been enhanced to include support for polygons and for bitmap elements. A bitmap element is a rectangular entity which contains a bitmap image. Typically bitmap elements are added to graphics objects by cutting a portion of a bitmap image object and pasting it into a graphics object (see Section 4.7.5). Bitmap elements are often useful as a background for a drawing.

In addition, means have been provided to control the way objects are aligned with respect to one another. For example, it is possible to align the tops, bottoms, right side, left side, or centers of a set of specified elements in a graphical drawing.

4.7.5 Clipboard

The notion of a "clipboard", similar to that found in the Apple Lisa and MacIntosh systems, has been added to EditDoc. It is now possible to cut an item (either a whole object or part of an object) from one document and paste it into another (or the same) document.

Unlike the Lisa and MacIntosh implementation of clipboards, the number of items that a Diamond clipboard can hold is determined by a user preference setting. The clipboard is managed as a stack. Items that are cut are "pushed" onto the stack. When an item is cut and the stack is full, the oldest element on the stack is discarded to make room for the new item to be added. When pasting an item into an object, if the desired item is not on the top of the stack, the user may cycle . .rough the clipboard stack until the desired item is reached.

The cut and paste operations work both within a single given data type (e.g., cutting from a graphics portion of one document and pasting into a graphics portion of another) and between two different data types (e.g., cutting from a graphics portion of one document and pasting into a bitmap portion of another). Of course, when an object of a given type on the clipboard is pasted into an object of a different type, conversions are performed on the source object to make it conform with the destination object. In some cases this may be infeasible; for example, converting an image into text.

4.7.6 Miscellaneous

New commands have been added to EditDoc which enable a user to see the "next" or "previous" document in the "current" folder. This makes it easy to read a set of documents in a folder one at a time without having to explicitly close the current document and open the next (or previous) document. This is particularly useful when reading new messages from the InBox folder.

4.8 Import / Export Manager

The Import/Export Manager is responsible for exchanging multimedia messages with other multimedia systems. To send a message, it first translates the message into a standard format for transmission, and then sends the message. The transmission format is part of the emerging DARPA multimedia protocols.

At the July meeting of the DARPA multimedia message system community, an agreement made to between ISI, SRI, and BBN to convert from RFC 759/767 format for multimedia message exchange to a variation of that format specified by SRI. It was agreed that the conversion should be completed by October 1, 1984. We have modified our Import/Export Manager to transmit the new format and to receive either format.

The new multimedia format specified by SRI is deficient with respect to text protocols. It does not allow nested structures (like an enumeration within an itemization within an enumeration). We have designed an extension to the format and the software to implement that extension. We have not implemented the extension yet. The format is also deficient in not allowing font specifications.

The Import/Export Manager is the only major component of the Diamond system that has not yet been ported to the Sun Workstation. The major problem is that the current Import/Export Manager uses the BackgroundServer and background FTP requests to receive/deliver messages from a remote MPM (we are using ISI's MPM). Most of the manager is concerned with translating between formats, and that part will port to the Sun easily. The delivery mechanism will not port easily. Our current plan is to make use of the Sun implementation of the MPM developed at SRI (which is not yet available), and modify the delivery part of the Import/Export Manager to use the MPM on the same machine.

4.9 Papers, Documentation, Meetings, and Presentations

During this period, we developed the following material for describing the Diamond system:

- A new video tape describing the goals, architecture, hardware base, user interface and use of Diamond.
- "Future Workstations and Applications", H. C. Forsdick, NOAA Conference on Future Directions in Data Communications, Denver, CO, July 1984.
- "The Diamond Multimedia Message System", H. C. Forsdick, Federal Computer Conference, Washington, D. C., September 1984.
- "Diamond: A Multimedia Message System Built Upon a Distributed Architecture", R. H. Thomas, H. C. Forsdick, T. R. Crowley, G. G. Robertson, R. W. Schaaf, R. S. Tomlinson, V. M. Travers, to be published in IEEE Computer, Summer 1985.
- "The Diamond Multimedia Document Editor: User's Guide", BBN Report No. 5724, (DRAFT) July 1984.
- "Multimedia Mail Meeting Notes", H. C. Forsdick, Minutes of a meeting about experimental multimedia message systems held at BBN on 23-24 July 1984, NWG RFC 910.

In addition, we attended the IFIP 6.5 Working Conference on Computer Message Systems held at Nottingham England and presented a paper on Diamond titled "Initial Experience with Multimedia Documents in Diamond". The paper was published in the conference proceedings, Computer-Based Message Services, Hugh T. Smith, editor, North Holland, 1985. A slightly revised version of this paper was also published in the September 1984 issues of the IEEE Quarterly Bulletin on Database Engineering.

5. THE JERICHO JADE SYSTEM

5.1 Image Manipulating Operations

Fast algorithms for scaling and rotating bitmap images by arbitrary amounts have been developed, and routines which implement these algorithms have been added to the Jade ImageOps library. The EditDoc multimedia editor makes use of these image manipulating routines to permit users to rotate and scale the image components of multimedia documents (see Section 4.7).

Image scaling is done by adding or removing rows and/or columns of the bitmap image to achieve the desired size image from the original image. When a row or column is added, it is a duplicate of the adjacent row or column. When a row or column is removed, none of the adjacent rows or columns are adjusted. The added (removed) rows or columns are distributed evenly over the extent of the image so that effect of enlargement (reduction) is applied to the entire image. Figure 4 illustrates how this is done. When an image is scaled in both dimensions, the scaling is first done in one dimension and then in the other. In interactive image editing operations, to preserve as much accuracy in scaled images, we always start from the original image and scale up or down from it. Figures 5 and 6 show examples of an image that has been enlarged and reduced. In one case the aspect ratio of the original has been preserved by forcing the scaling in the horizontal dimension to be a linear function of scaling in the vertical dimension. In the other case there no linkage enforced on scaling in the two dimensions.

The algorithm for rotating bitmap images through arbitrary angles assumes that an efficient means exists to transfer an arbitrary rectangular region of a bitmap image from one location to another. Also assumed is a method of rotating images through multiples of ninety degrees.

The rotation algorithm is based on the fact that a rotation transformation can be decomposed into a sequence of three shear transformation such as the following:

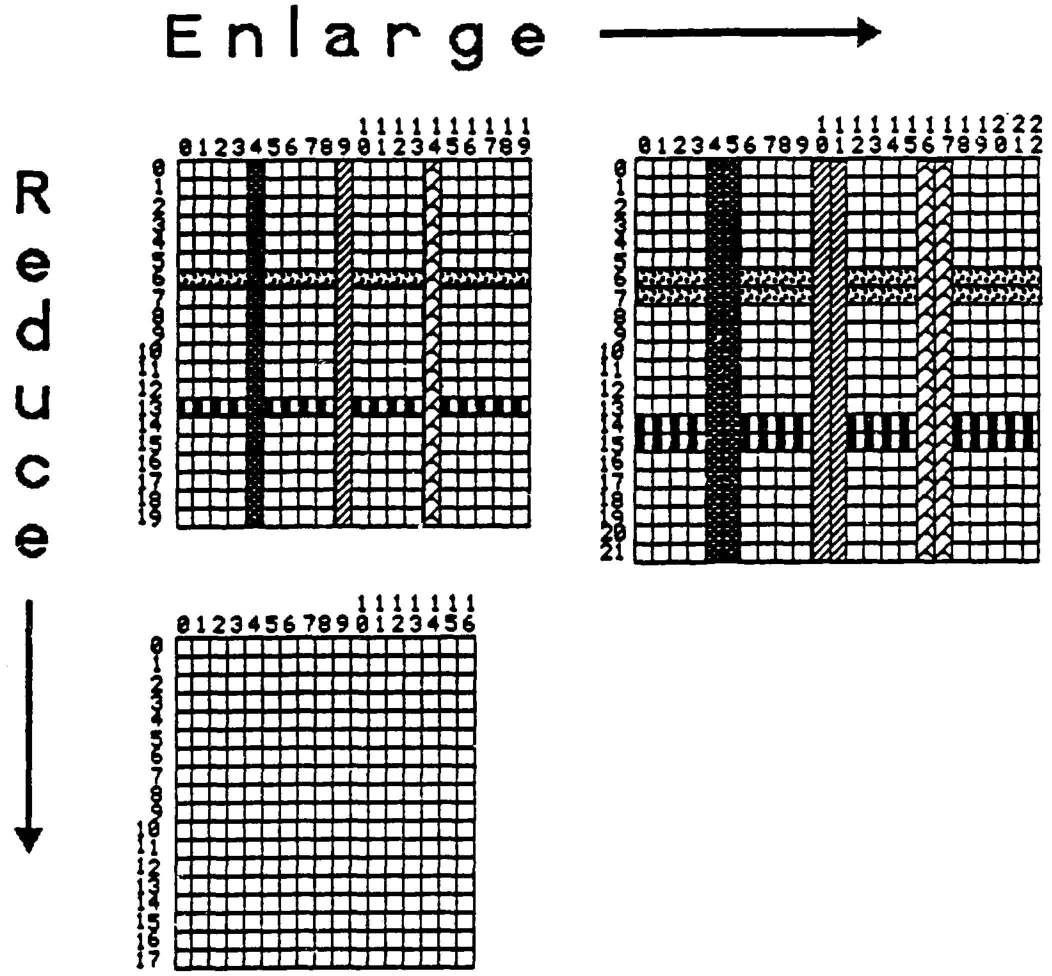


Figure 4. Scaling of BitMap Images

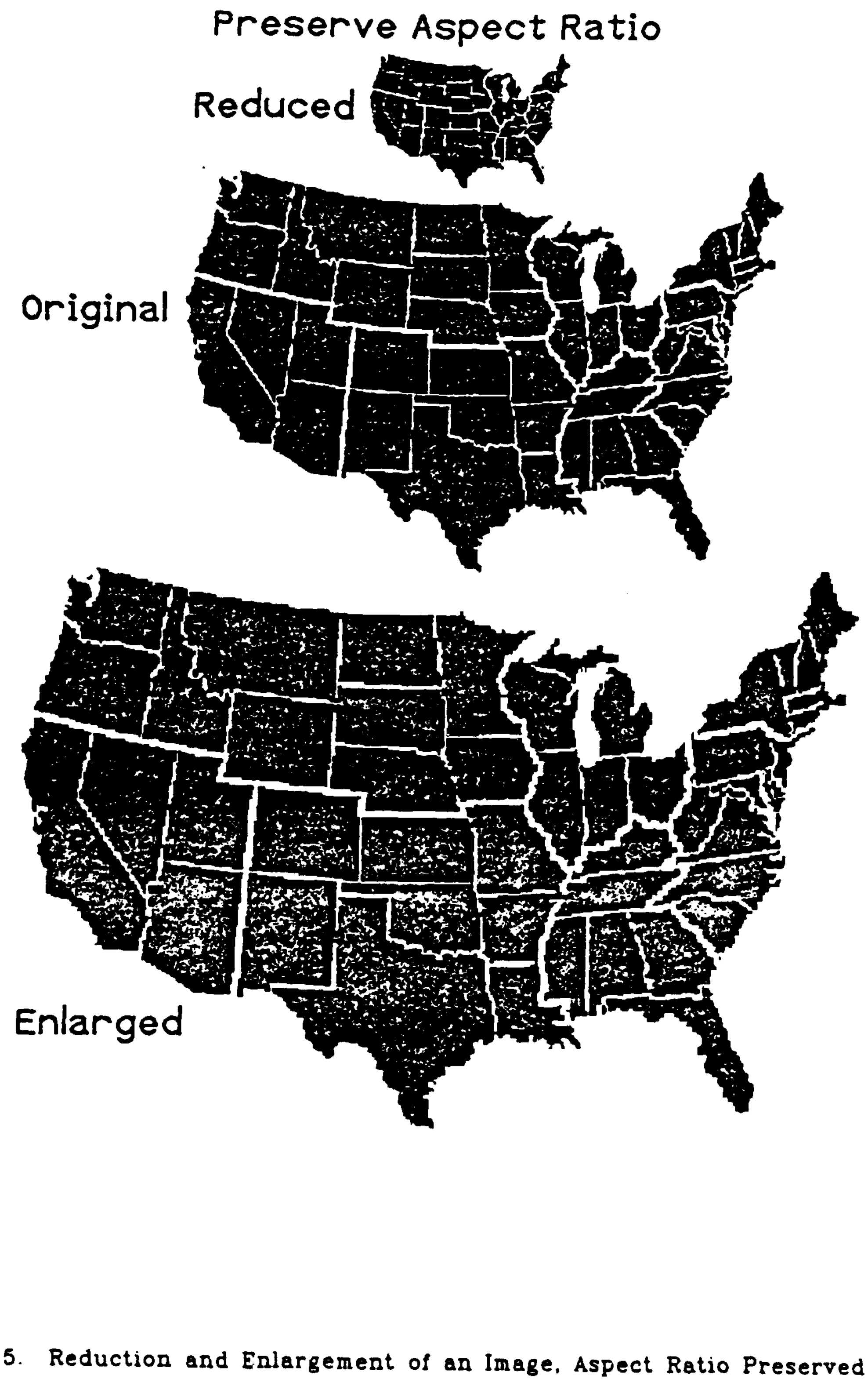


Figure 5. Reduction and Enlargement of an Image, Aspect Ratio Preserved

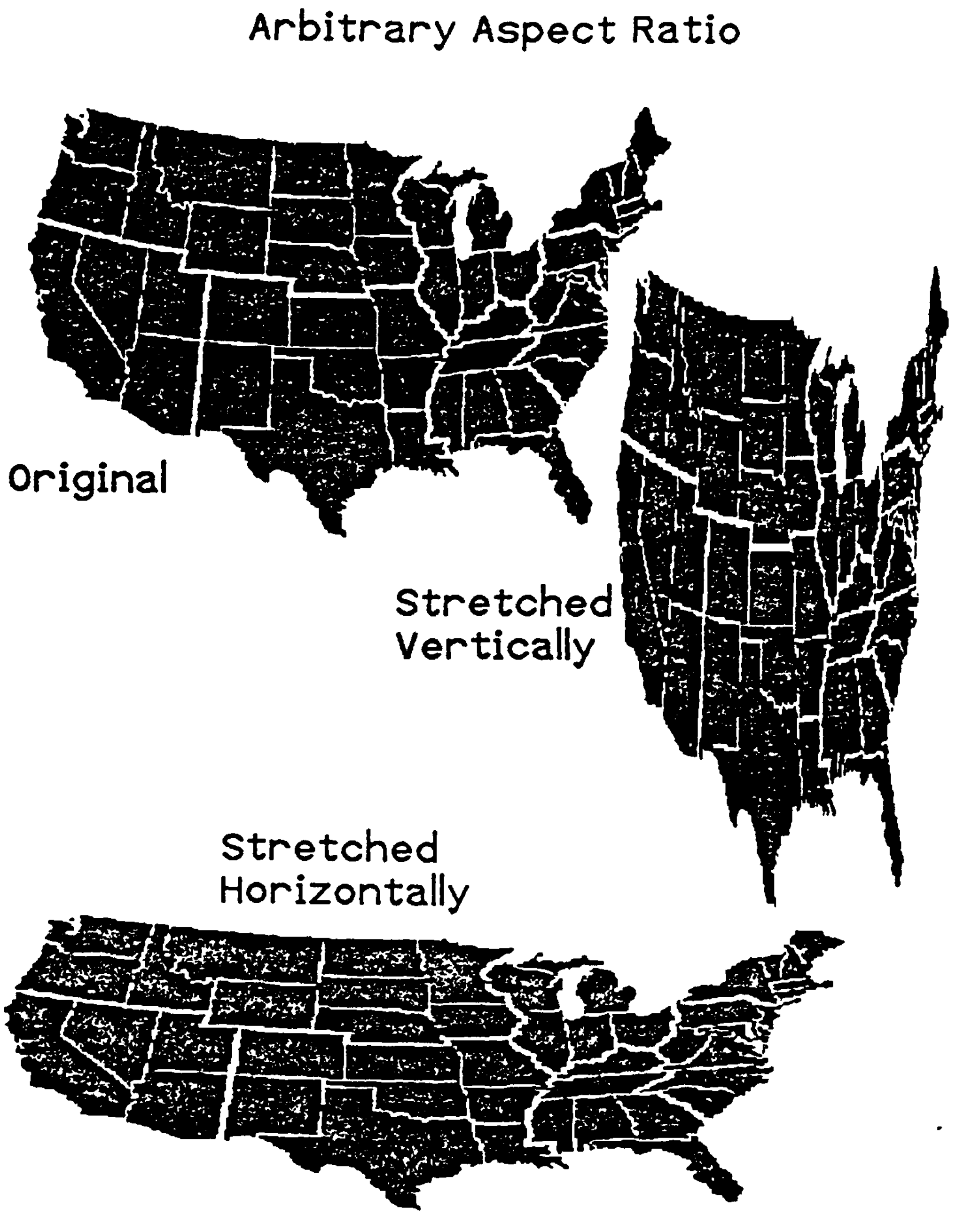


Figure 6. Reduction and Enlargement of an Image, Arbitrary Aspect Ratio

$$R = A * B * A$$

$$\begin{pmatrix} \cos(x) & \sin(x) \\ -\sin(x) & \cos(x) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} * \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix}$$

Where $a = (\cos(x) - 1)/\sin(x)$ and $b = \sin(x)$.

Transformations such as shearing and rotation are mathematical operations which cannot be implemented exactly when the coordinates of points must be integers. Thus the implementation of a shearing operation is faced with the same problem as the implementation of a line-drawing operation. The solution is the same: the transformed points are moved to the nearest (in some sense) available raster position. The effect of this on a horizontal shearing operation is illustrated in Figure 7.

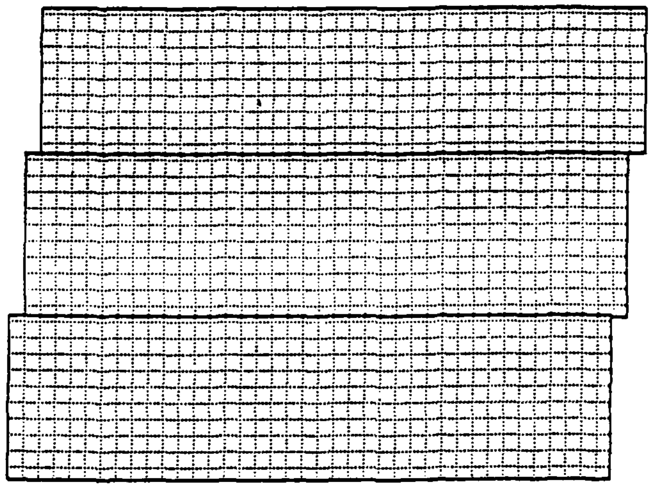


Figure 7. The effect of a horizontal shearing operation.

In fact, the necessity for moving pixels by an integral number of raster positions is precisely the reason that this algorithm is efficient; many pixels must be moved by the same amount and we have postulated the existence of an efficient means of doing this. All pixels in a given row move by the same amount and, furthermore, several adjacent rows also move by the same amount. The smaller the angle of the shear is, the larger the size of the block is. As the angle approaches 45 degrees, the block become less high until at 45 degrees, each block is just one pixel high. The picture for vertical shear is similar except, of course, for a change of axes.

For maximum efficiency, as many pixels as possible should be moved in each operation. This means that for angles from 0 to 45 degrees, the first shear should be horizontal and for angles from 45 to 90 degrees, the first shear should be vertical. For angles outside the range of 0 to 90 degrees, other algorithms should be employed to rotate by +90, 180, or −90 degrees followed by the present algorithm.

It is also necessary for the discontinuities in the first and last shearing operations to be properly positioned. This is especially true for small angles. The reason is that if the discontinuities in the two shearing operations occur at the same places, adjacent pixels which straddle the discontinuity will end up being separated by two positions. To avoid this problem, the discontinuities are displaced so as to be maximally separated at the middle of the picture.

Figure 8 illustrate the effect of rotating a bitmap image through a small angle. (Notice that the rear edge of the backpack has been made vertical.) Because this image started with a fairly large amount of noise in it, the distortions due to the rotation process are not very noticeable. Images with greater regularity or finer features are more noticeably distorted.

5.2 Compatibility with Sun Workstations

The execution environment provided to application programs by the Jericho Jade and Sun Unix systems differ in two ways which have had a significant impact on porting applications from Jericho to the Sun:

1. Jericho permits processes to share address spaces, and the Sun does not;
2. The Jericho window system automatically repairs all "damage" done to a

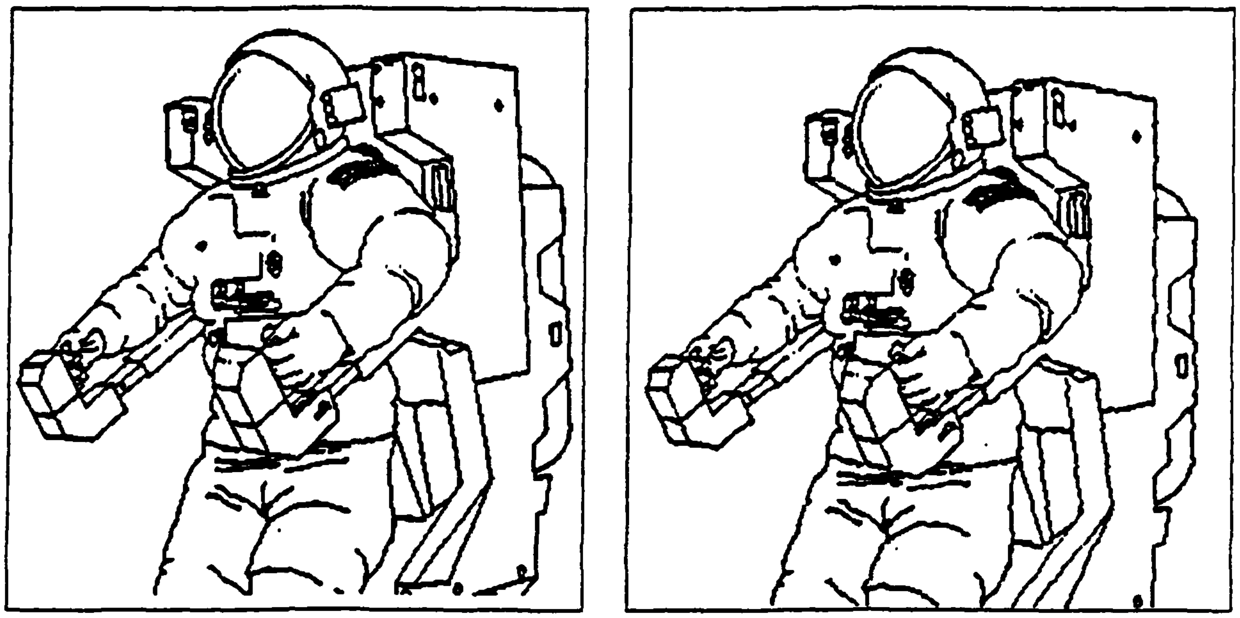


Figure 8. The effect of rotating an image through a small angle.

window when the window is unburied[3], and the Sun does not. Rather it passes a list of the damaged areas of the window to the application program which is expected to repair the damage.

This section summarizes how we dealt with these differences in porting Diamond to the Sun.

Our goal at the outset of the porting effort was to maintain a single set of sources for Diamond which could be compiled either for Jericho or for the Sun. Diamond makes extensive use of the Jade program libraries. Furthermore, many of the places where changes had to be made to handle the differences between Jericho and the Sun were in the libraries. Consequently, to meet our goal of a single set of sources it was necessary to audit the Jade software and, where necessary, to retrofit it to be compatible both with the Sun and Jericho environments.

[3]Those parts of the window that were under another window are considered "damaged" and need to be "repaired" by restoring their current contents.

The fact that Sun processes do not share memory forced a number of substantial changes throughout Diamond. The database mechanisms used by the Document Store Manager and the Authentication Manager, and the document store cache mechanism used by the Access Point (see Section 4.4) made extensive use of shared memory. These mechanisms have been redesigned to use a Database server process (see Section 4.1.2). The Coordinator component of the Access Point made use of shared memory to control the set of processes used to implement various Access Point functions. That control mechanism had to be redesigned to operate without shared memory. Various Diamond components use a synchronization mechanism to coordinate their internal activity. This synchronization mechanism was implemented on Jericho using shared memory in order to achiever maximum efficiency. The implementation of the synchronization mechanism was completely redesigned to make use of a Synchronization Server process, while the interface to it provided for application programs was preserved.

Differences in the window systems between Jericho and the Sun were handled in two ways:

1. The Jade WindowLib library, which is the program library application such as Diamond use to manipulate display windows, was ported from Jericho to the Sun. This provided an application interface to window manipulating functions for the Sun that was identical to that on Jericho. Consequently, the parts of a program that do routine window manipulations do not have to be modified when the program is moved from Jericho to the Sun.

2. The notion of "window hints" was expanded to include a new "window has been damaged" hint[4]. This new hint makes it possible for an application to discover when one of its windows has been damaged and to take appropriate action to repair it.

In order to make various parts of interactive applications, such as the Diamond Access Point, responsive to window hints (damaged and changed hints in particular), all user input functions (i.e., uses of pop up windows, menus, forms, scrolled objects, and the Access Point library) must be prepared to return control to the application when an unexpected message (i.e., any non-character, such as a window hint) arrives. During this reporting period, we modified all of our basic user interface libraries and the various parts of the Diamond Access Point to deal with these hint messages

[4]A window hint is a signal sent to a process by that window system that an event relevant to a window of interest to that process has occurred. See Semi-annual Technical Report No. 5 for more details.

correctly.

This set of changes was substantial, but was forced on us because of the nature of the Sun window system, which does not maintain off-screen images of partially buried windows. When a partially buried window is exposed, damaged hints are sent to the process owning the window, and that process is expected to repair the damage. The problem for an application, such as the Access Point, is that these damage hints may arrive when some very low level user interface function, such as the one that pops up a menu and gathers the menu item specified by the user, has control. The low level functions have no notion of what is actually on the screen. Hence they have to propagate the hint up, possibly through several layers, to the application, which can then reconstruct the display to repair the damage. Since Pascal does not have an exception mechanism or the ability to pass procedures as parameters, in porting the Access Point, we were forced to restructure all the layers of the user interface to make it possible to pass these hint messages back up to higher levels, and then, after the hints have been handled, to return to the lower levels without losing any context.

6. THE JADE PROGRAMMING ENVIRONMENT

6.1 Network protocols and IPC

The interhost interprocess communication facility has been successfully ported to the Sun. Moving the IPC facility required some minor changes to both the IPC and the underlying TCP/IP protocol support. These changes are described below.

The Sun TCP implementation supports the notion of "continuous listening" ports. As the name suggests, a continuous listening port is one which remains in the listening state at all times, even while a connection to the port is being established. Without continuous listen the application process using the port must "recycle" it into the listening state after a connection attempt is handled. This means that there is a (usually brief) period of time during which the port is not in the listening state, and during which attempts to connect to it from remote processes must fail. With continuous listening, attempts to connect to the port while another connection attempt is in progress will not fail due to absence of the listening port. Sir :e listening ports are typically used to support "services", the continuous listening port mechanism makes sure that the service is continuously accessible from the network.

The Jade TCP implementation, which previously did not support continuous listening ports has been augmented to include the notion, and the Jade TCP interface library for both the Jericho and the Sun has been extended to explicitly support the the notion of continuous listening ports.

In addition, the IPC implementation has been changed to use continuous listening ports. This was done so that the IPC manages its TCP connections the same way on the Sun and the Jericho. Before this change, the IPC was written to explicitly recycle its listening port. The IPC uses a continuous listening port, and no longer explicitly recycles the port. In addition, the IPC handled failures to connect to other IPCs by immediately retrying a few times in order to deal with the possibility that the failure was due to "dead time" at the remote IPC while its listening port was being recycled. This was not changed, because the IPC may need to communicate with other IPCs which must explicitly recycle their listening ports.

The ability to broadcast messages to all hosts in a Diamond configuration plays an important part in the Diamond implementation. In particular, it is used by the IPC to locate objects[5].

With the addition of Suns on an Ethernet, the Diamond development configuration now is distributed across two local networks, the Jericho Fibernet and the Sun Ethernet, which are connected by a gateway. We expect that future Diamond configurations will also span several local networks. For Diamond to work with such a configuration the ability to broadcast must be extended to span the networks which comprise the configuration. To deal with the short term problem of two networks connected by a gateway, we have modified the Jade IP implementation so that packets broadcast on the Jericho fibernet are propagated to the Sun ethernet, and have modified the interface to the Sun IP to that. We expect that it will be necessary to refine the broadcast propagation mechanism as the topologies of configurations become more complex.

In experimenting with configurations spanning two networks we encountered a problem due to the apparent inability of the Sun network code to reassemble more than 2 fragments of an IP datagram back into the datagram. The problem was discovered while sending IPC small messages from a Jericho through a gateway to a Sun. The maximal size of an IPC small message, exclusive of headers, is 1240 bytes. Small messages are typically sent as single IP datagrams. The Jericho IP implementation had been set up to fragment IP datagrams into 576 byte fragments for transmission through the gateway that connects the Jericho Fibernet to the Sun Ethernet. We discovered for IPC small messages sent from a Jericho to a Sun, that messages fragmented into two or fewer fragments were properly received by the Sun but those fragmented into three fragments were not. This problem was corrected by "reconfiguring" the Jericho IP to fragment IP datagrams into larger fragments so that a maximal IPC small message would be broken into at most two fragments.

6.2 Software State Database

During the past six months, the Software State Database system (hereafter referred to as SoftwareState) has become a regular part of our development environment. Distribution of new software is done exclusively through the mechanisms provided by SoftwareState. In addition to improving the robustness of the original

[5]See Semi-Annual Technical Report No. 4, BBN Report No. 5722 for details.

operations, a number of significant improvements have been made to the system. These include:

- The inclusion of dependencies between files into the database.
- The use of dependencies to check the consistency of an operation.
- Separate create dates for each file system being maintained by SoftwareState.
- Support for working on a large system in a private environment.
- Integration of the Sun workstations into SoftwareState.

6.2.1 Dependencies

Within a complex software environment, dependencies naturally arise between modules. Object files depend on the source files they are compiled from and on the interface of the libraries they use. In our environment, assembly language files for the Sun depend on the object files they are translated from.

When a file is Installed, SoftwareState analyzes it to automatically determine which files it depends on. These forward dependencies are entered into the database along with the symmetrical backward dependency. (That is, the database explicitly contains the fact that object A uses B and the distinct but related fact that B is used by object A.) These dependencies may be perused by a user or used by the system to check the consistency of a requested operation.

SoftwareState recognizes three different kinds of dependencies:

- **Time.** A time dependency implies that the dependent file must have been created at a later time than the file it depends on. An object file has a time dependency on the source file it is compiled from.
- **Interface.** An object file has an interface dependency on a library which it uses. The object file must be recompiled if the interface changes, but does not need to be recompiled if only the implementation part of a library changes.
- **Simple.** A simple dependency is used to break a loop in an interface dependency cycle. It is intimately related with the compilation strategy used in the Jade environment to handle dependency loops within a set of libraries.

Pascal object files and Sun assembly files are analyzed to determine the files they use. Dependencies which cannot be derived automatically could be entered into the database by hand. There are several complications with regards to dependencies.

- **Interfaces.** In the Jade environment, the implementation of a library resides in the same file as its interface. In SoftwareState, we wish to be able to distinguish when only the implementation of a library has changed (thereby recognizing that programs which use the interface do not need to be recompiled). The write date of the file does not contain enough information.

  To deal with this problem, the SoftwareState system maintains an *Interface Write Date*. SoftwareState calculates a value for the interface of a library when it is Installed. This value is a single integer derived from a CRC of the characters in the interface. When a new version of a library is Installed, SoftwareState recalculates the CRC of the interface. If the CRC value has changed, then the library gets a new Interface Write Date. Other files which use the changed library will need to be recompiled. A programmer or tool can determine what recompilations are necessary by comparing the write date of an object file with the interface write date of the libraries which it uses.

- **Dependency Loops.** The simplest example of a dependency loop is a library L1 which uses a library L2 in its implementation. Library L2 in turn uses library L1 in *its* implementation. This creates a problem in the Jade environment because there is no acceptable compilation ordering. L1 must be compiled before L2 and L2 must be compiled before L1. In order to get around this problem, it is possible to specify that library L1 uses the interface contained in the source of L2 rather than in the compiled version of L2. L1 can then be compiled before L2.

  The consequences of this for the dependencies contained in the database are as follows:

  - Compiled library files typically have interface dependency on other compiled library files (L1.Library depends on L2.Library). When a library explicitly uses the interface in the source file of another library than the interface dependency is on the source file (L1.Library depends on L2.Pascal).

  - Despite the fact that a library L1 is compiled using the source version of another library's interface, when it is run it will use the compiled version of that interface. To reflect this fact, a *Simple* dependency is added from L1.Library to L2.Library. This ensures that a check to see if all the dependencies for L1 are satisfied will also check all the dependencies for L2.

6.2.2 Using Dependencies

The simplest use of dependencies is to make them available for viewing. A programmer contemplating a change to the interface of some library can look in the database and find *every* file which uses it. This is especially useful if the interface change requires editing the dependent files rather than simple recompilation.

SoftwareState checks dependencies on CheckOut, CheckIn and Install. When some constraint is not met, SoftwareState will warn the user. However, a user may ignore the warning and proceed with the operation. For example, it is possible to Install a new version of a file on the Jericho without first creating and installing the dependent Sun assembly file. This allows us to experiment with some new feature on the Jericho before going through the additional overhead of creating a version for the Sun. After experimenting, we can go back and determine what needs to be done to bring all files up to date.

The constraints for each operation are described below.

- **CheckOut.** SoftwareState checks to make sure that all files which have time dependencies on the files being checked out are included in the set being checked out. The simplest example is requiring that the user check out the object file along with the source of a library or program.
- **Install.** Install checks time and interface constraints. This is to prevent the release of incompatible versions of interfaces and the programs which use them. It will also catch the failure to compile some source file.
- **CheckIn.** CheckIn also checks time and interface constraints. This is because a user may wish to Install some version for experimental use, but a CheckIn operation presumes that the programmer is finished with the file and should ensure that all constraints are satisfied.

8.3 IPC Monitoring Facility

The IPC monitoring facility is a tool which makes it possible to observe interactions among components of a distributed application, such as Diamond. The interactions are recorded by the monitor as the monitored application executes, and can be observed by a user in (near) real time or can be repeatedly played back to study operation of the application. The monitor understands both the higher level protocols used to support interactions among components of distributed applications and the lower level protocols. This makes it possible to parse and display the contents of interprocess messages as well as to display the patterns of message interactions. Because the execution of a distributed system, even over a relatively short period of time, can involve a large number of interactions, the IPC monitoring facility supports a very flexible "message filtering" capability through which a user can specify those interactions that he is interested in observing. We expect that the IPC monitoring facility will be extremely useful in the debugging and demonstration of distributed applications.

The IPC monitoring facility is composed of the following four types of component processes:

- o **IPCMonitor** - This process supports the user interface to the IPC monitoring facility. Through this interface, a user may add and remove monitoring filters, enable or disable monitoring on the distributed host computers, and control the display of monitoring information that is being collected and retrieved by other components of the IPC monitoring facility.

- o **MetaMsg Manager** - This process runs on the same host as the IPCMonitor process and manages the communication between that process and the MonitorData manager processes that are distributed throughout the local area network. The MetaMsg manager distributes commands from the user interface (e.g. enable monitoring, add filter) and receives the monitoring information that is being collected by the remote IPCServer processes. This monitoring information is received in the form of meta-messages which are recorded in a log file and are forwarded upon demand to the IPCMonitor process so that the information that they contain may be displayed to the user.

- o **MonitorData Manager** - Just as the MetaMsg manager manages the communication between an IPCMonitor process and the distributed MonitorData manager processes, the MonitorData manager process manages the communication between a local IPCServer process and all of the MetaMsg manager processes that exist in the local area network.

- o **IPCServer** - The main responsibility of the IPCServer is the transport of interhost messages. A portion of the IPCServer, however, is devoted to supporting the IPC monitoring facility. This support entails managing a data base of active monitoring filters, matching messages against these filters in order to determine whether a message is being monitored, and sending information about the monitored messages back to the monitoring program.

As of the end of the last reporting period, the initial design of the IPC monitoring facility had been completed along with initial implementations of both the MonitorData manager component and the IPCServer modifications to support monitoring.

During this reporting period, the initial implementations of the IPCMonitor and MetaMsg manager components were completed and the first version of the IPC monitoring facility was released. For the remainder of the reporting period since the monitor's release, our work has focused on improvements to the program's performance and enhancements to its user interface.

6.3.1 IPC Monitor Scenario

This section presents a series of figures that illustrate the use of the IPC

Monitor. The IPC Monitor maintains a dynamic display which shows interactions among the components of the distributed application being monitored. The figures presented are software "snapshots" of the display as a short Diamond scenario was being monitored. The scenario chosen is one in which Diamond retrieves a document from one of its document stores. This involves retrieving the DocStruc for the document (i.e., the data structure which defines the document) and all of the atomic objects referenced by the DocStruc.

Figure 9 shows the display of the IPCMonitor program, immediately after the document retrieval has started. The three circular objects represent host computers. The *Schaaf* host is shown at the top of the display with the *Jade* and *Diamond* hosts being shown in the lower left and lower right, respectively.

The ovals shown inside the hosts are processes involved in running the Diamond application. The lines drawn between processes and hosts indicate currently active communication paths. Note that there does not currently exist an active path between the Jade host and the Diamond host.

The shaded ovals represent processes on the path of a message that is currently in transit. In this illustration, a message is in transit from the *EditDoc* process on the Schaaf host to the *DocStore* process on the Jade host. The message itself, is represented by the small circular icon containing an "S" shown next to the DocStore process on Jade. The "S" indicates that the message is a *small message*, and is therefore contained in a single network packet. Other types of messages of large messages ("L") and IPC protocol messages ("I").

The area beneath the hosts is used for the monitoring program's control switches and status indicators. The *Record* switch is used to request the display of monitoring information in (near) real time as the program being monitored executes. As this information is displayed, it is also recorded in a log file. The *Play* switch is used to request play back of a log file. The shading for the play switch indicates that the monitoring facility is now in playback mode. The *Pause* switch is used to freeze the action in the area above the control switches and is shown to be on. The Single-Step switch is also on. This causes the monitoring facility to pause whenever a message in transit reaches its destination.

The *Speed* switch is used to set the rate at which the monitoring display cnanges. Running at a faster rate allows the monitoring facility to keep better pace with the executing application, but may be too fast for the user to follow. Accordingly, the user may adjust the rate to whatever is comfortable. The current setting is a little more than one-half of the maximum rate and is indicated by the degree of shading shown in the switch indicator.

The *Log Size* status indicator shows the current position of the monitoring program in the log file being played back. In record mode, this status indicator gives a rough idea of how far the monitoring facility has fallen behind execution of the program being monitored. Currently, a little more than two-thirds of the log file has already been replayed.

Figure 10 shows the contents of the message from the EditDoc process to the DocStore process. The message contents include the unique identifiers (UIDs) for the source and destination objects, the type of operation that the source process is requesting to be performed on the destination object, and the request identifier information that is associated with the message. This particular message is a request from the EditDoc process to perform a ReadValue operation on the indicated DocStruc.

As the message was in transit from EditDoc to the DocStore, the user caused its contents to be displayed by holding down the left button of the mouse, positioning the mouse cursor above the icon for the message, and then releasing the button. Holding the left button froze the action, and releasing it caused the message contents to be displayed. Clicking the right mouse button at this point would cause the message contents to be taken down and would unfreeze the display.

If a more detailed display of the message contents is desired, the user may click the left mouse button while the mouse cursor is positioned over the message icon. Doing this, changes the display to that shown in Figure 11 in which the various fields of the message are displayed. The name of the field is shown in the first column and the value of the field is given in the second. For structured values (i.e., records or arrays), an indication is given that the field value is structured. An example of this is the *MonitoringInfo* field in Figure 11. The components of a structured field may be displayed by asking for *More Detail* on that field. The type that is associated with each field may be displayed by requesting *Verbose* mode.

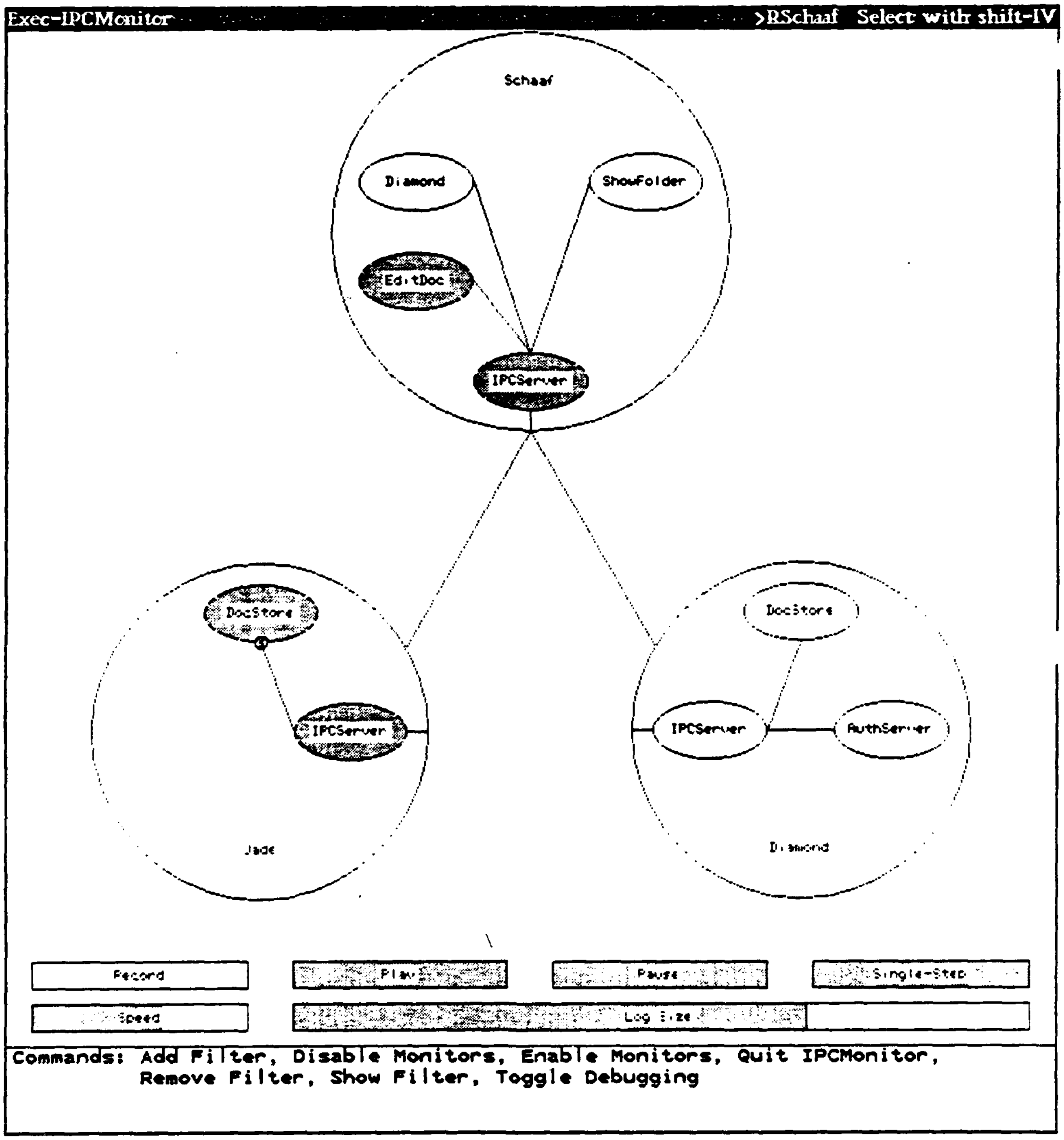
The request by the EditDoc process to perform the ReadValue operation on the

DocStruc cannot be satisfied until the DocStore process ascertains whether the EditDoc process is allowed to read the referenced DocStruc. The DocStore process checks this by sending an AuthBindingOf request to the AuthServer process on the Diamond host. Transmission of this message is shown in Figure 12. Note that as a result of this request, a communication link between the Jade and Diamond hosts is indicated.

In Figure 13, we see the response to the AuthBindingOf request from the DocStore process. Response messages are represented by an icon that is slightly different from the one used for request messages. Requests are represented by a white circle containing a black letter, while responses are represented by a black circle containing a white letter. The message contents for a reply is also different from that for a request in that the message contains reply code information. In this case, the reply code is SuccessComplete which indicates that the operation succeeded and that the EditDoc process is allowed access to the DocStruc whose value it requested.

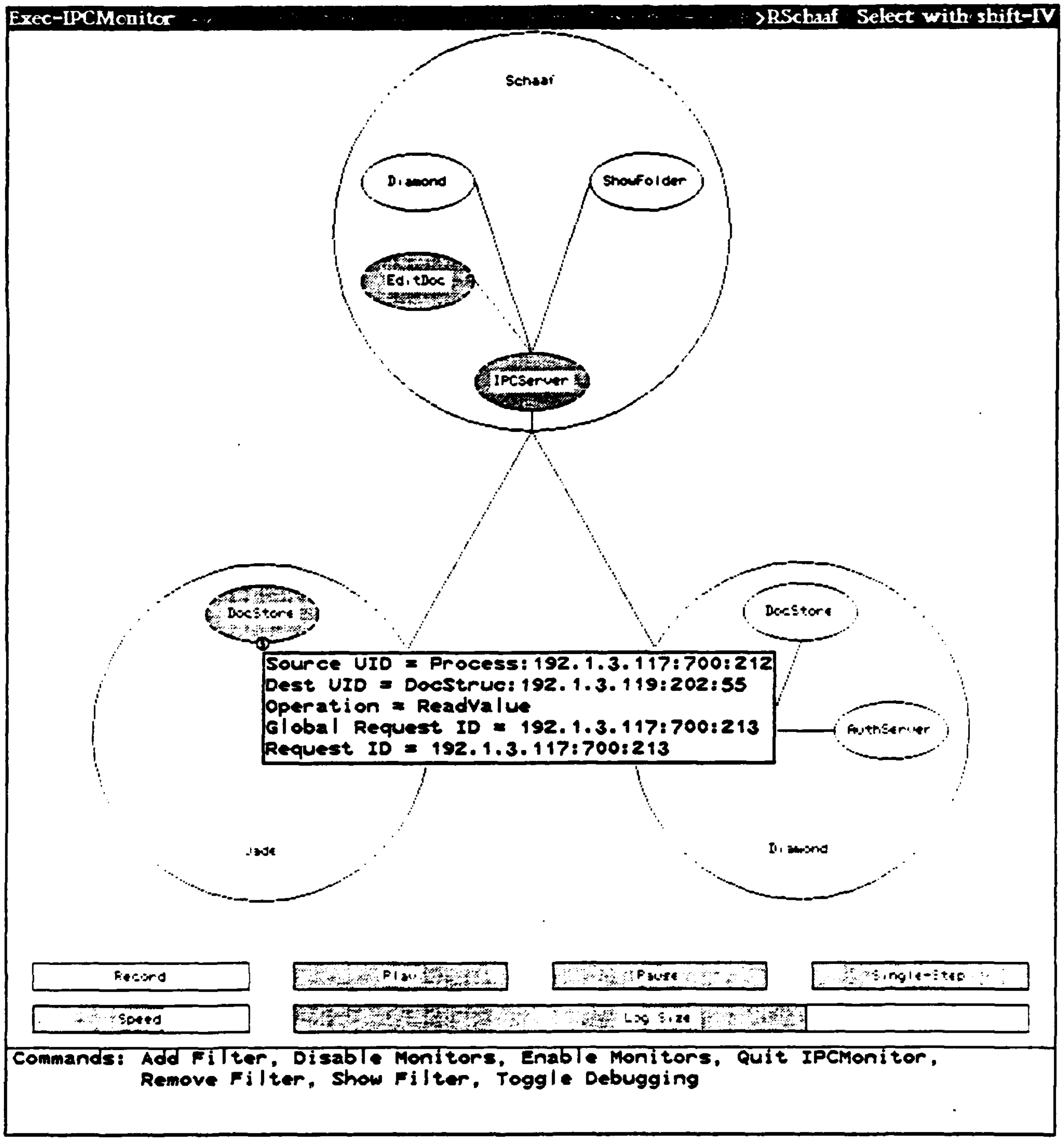
Figure 14 shows the transmission of the response to the original ReadValue request from the EditDoc process. Figure 15 shows the fields that are in this response message. This display was obtained by asking for more detail on the response message icon.

After the EditDoc process receives the value of the DocStruc object, it determines all of the atomic objects that are referenced by the DocStruc. These objects are then requested one at a time from the DocStore where they are located. Figure 16 shows a request from the EditDoc process to the DocStore process on the Jade host for a particular atomic image object. Upon receiving this request, the DocStore makes sure that the EditDoc process is allowed access to the object in the same manner that access to the DocStruc was checked. If the EditDoc process is found to have the proper authorization, the value of the atomic image object will be returned. This last step of showing the response containing the value of the atomic image is shown in Figure 17. Notice that the response is in the form of a large message, indicated by the "L" inside the message icon.



Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
Remove Filter, Show Filter, Toggle Debugging

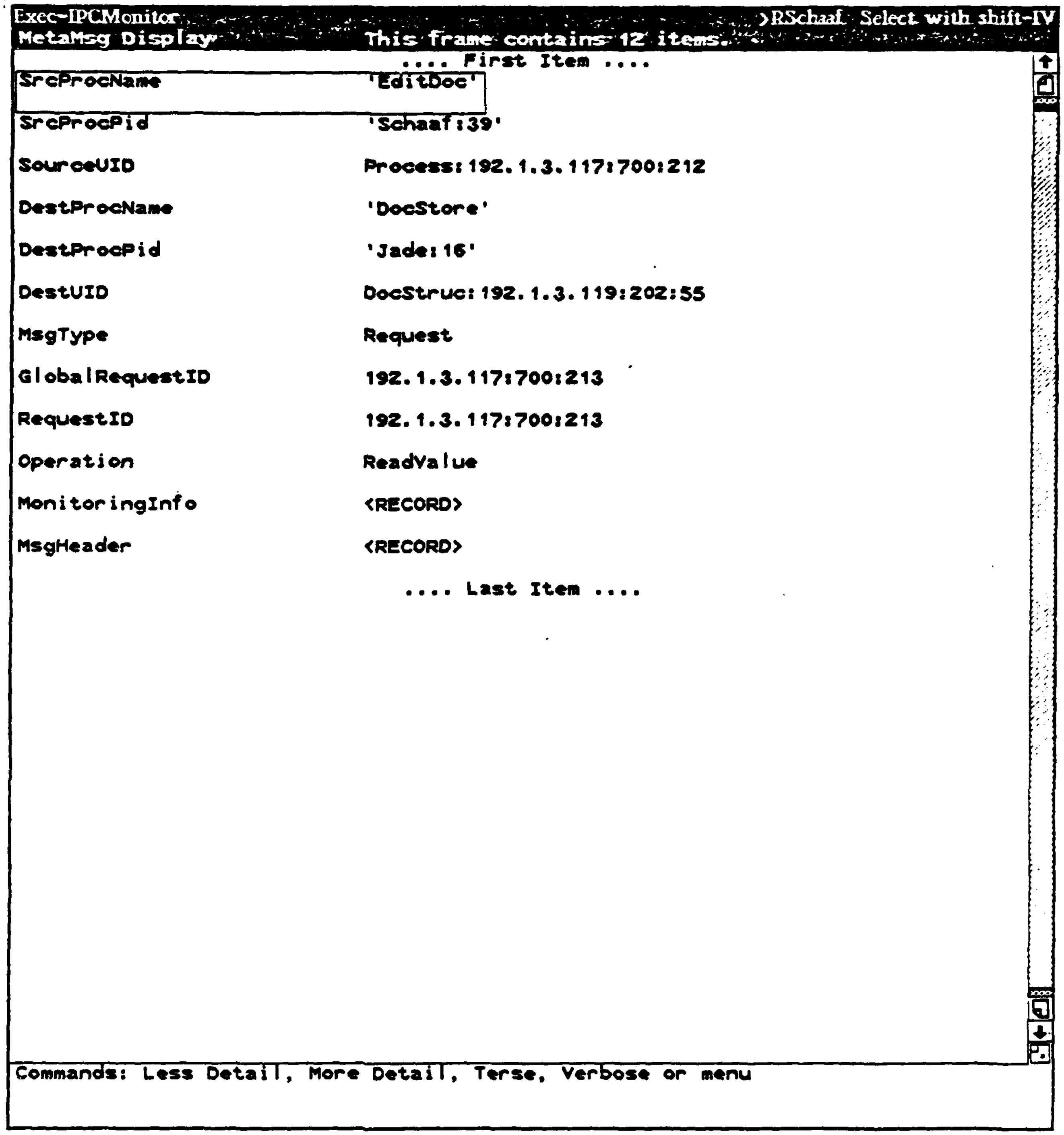
Figure 9. DocStruc ReadValue request



```
Source UID = Process:192.1.3.117:700:212
Dest UID = DocStruc:192.1.3.119:202:55
Operation = ReadValue
Global Request ID = 192.1.3.117:700:213
Request ID = 192.1.3.117:700:213
```

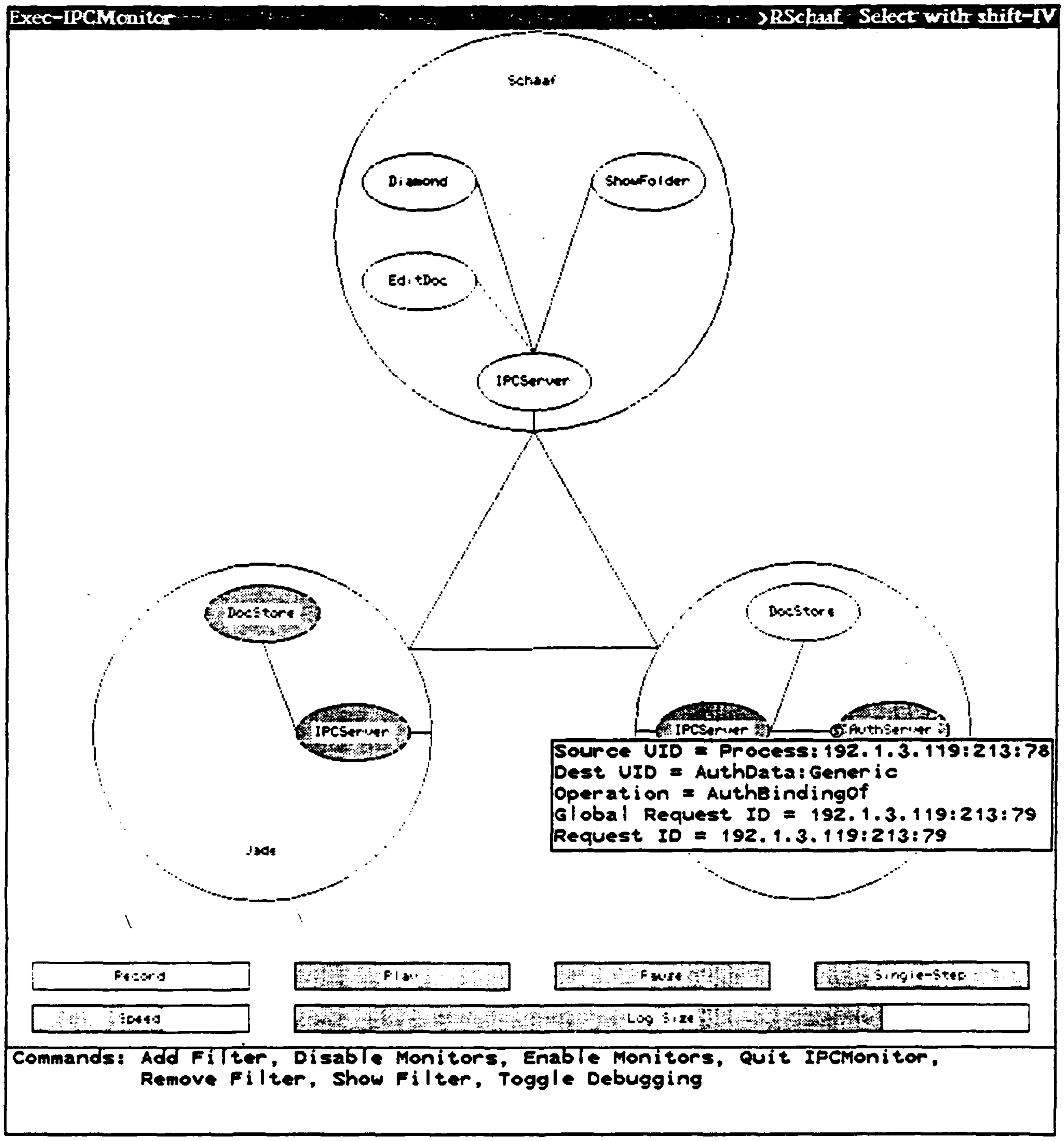
```
Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
          Remove Filter, Show Filter, Toggle Debugging
```

Figure 10. DocStruc ReadValue request - message description



```
Exec-IPCMonitor                                   >RSchaaf  Select with shift-IV
MetaMsg Display            This frame contains 12 items.
                         .... First Item ....
SrcProcName                  'EditDoc'

SrcProcPid                   'Schaaf:39'

SourceUID                    Process:192.1.3.117:700:212

DestProcName                 'DocStore'

DestProcPid                  'Jade:16'

DestUID                      DocStruc:192.1.3.119:202:55

MsgType                      Request

GlobalRequestID              192.1.3.117:700:213

RequestID                    192.1.3.117:700:213

Operation                    ReadValue

MonitoringInfo               <RECORD>

MsgHeader                    <RECORD>

                          .... Last Item ....

Commands: Less Detail, More Detail, Terse, Verbose or menu
```

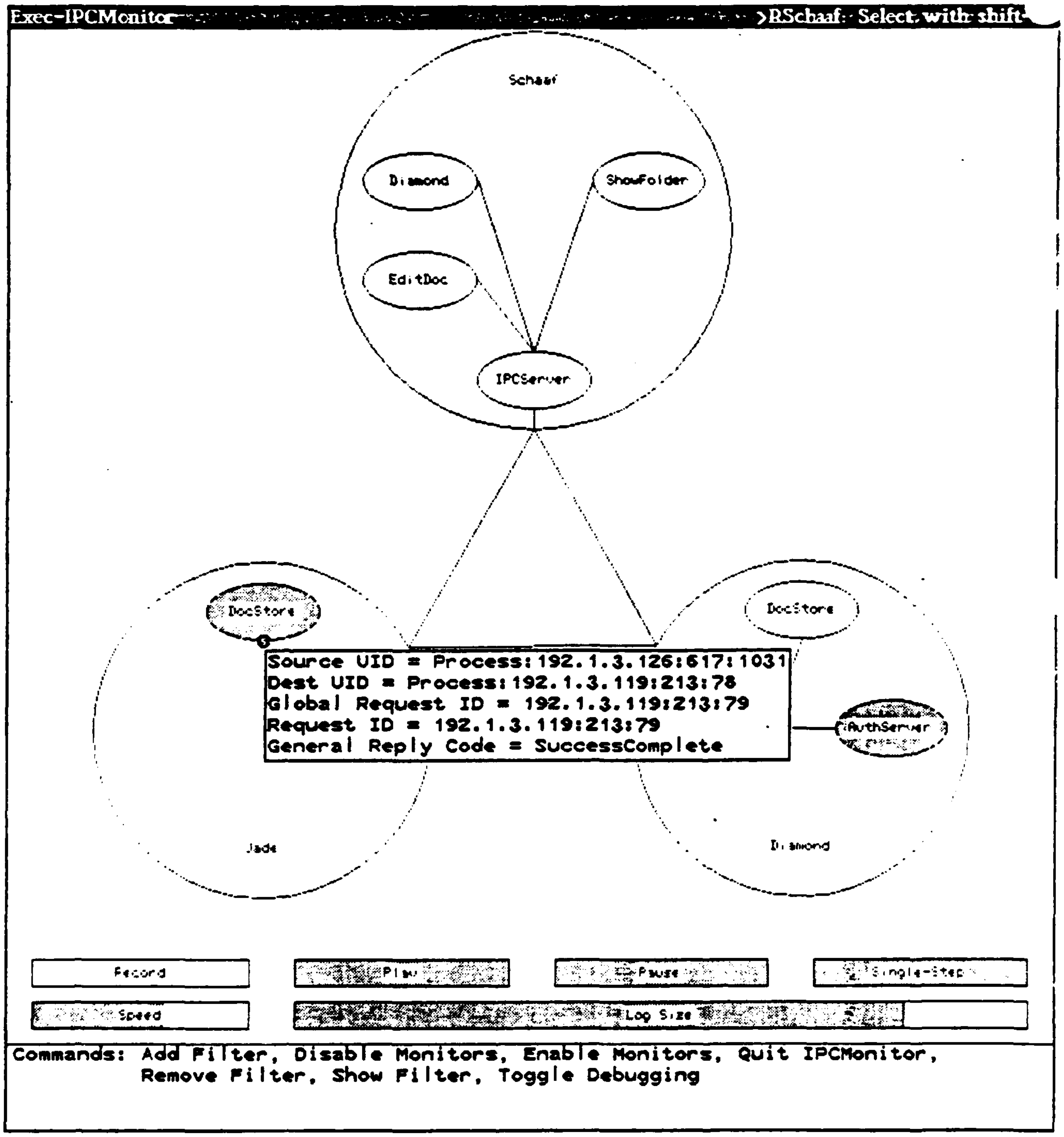
Figure 11. DocStruc ReadValue request - detailed information



```
Source UID = Process:192.1.3.119:213:78
Dest UID = AuthData:Generic
Operation = AuthBindingOf
Global Request ID = 192.1.3.119:213:79
Request ID = 192.1.3.119:213:79
```

```
Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
          Remove Filter, Show Filter, Toggle Debugging
```

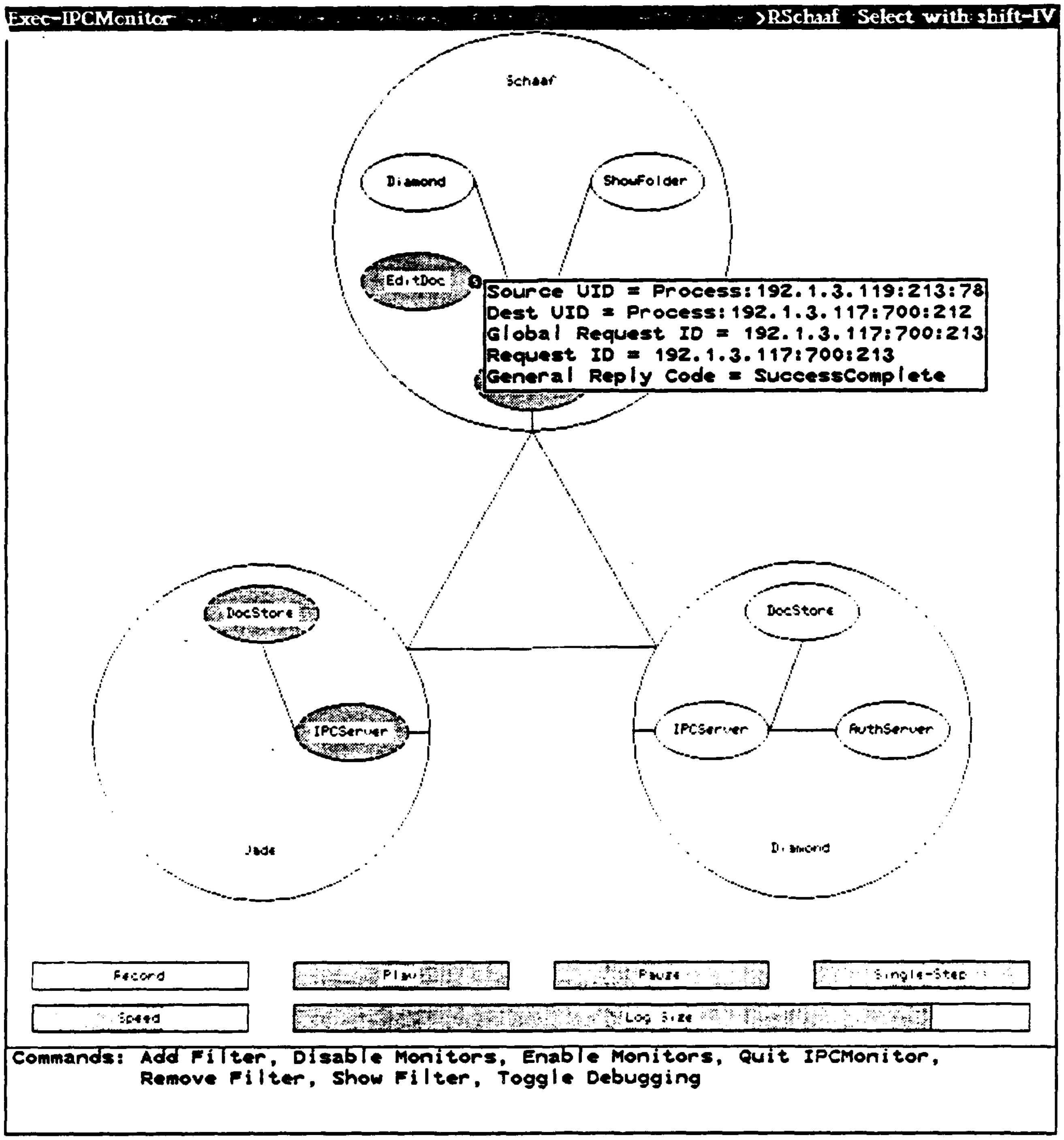
Figure 12. AuthBindingOf request



```
Source UID = Process: 192.1.3.126:617:1031
Dest UID = Process: 192.1.3.119:213:78
Global Request ID = 192.1.3.119:213:79
Request ID = 192.1.3.119:213:79
General Reply Code = SuccessComplete
```

```
Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
          Remove Filter, Show Filter, Toggle Debugging
```

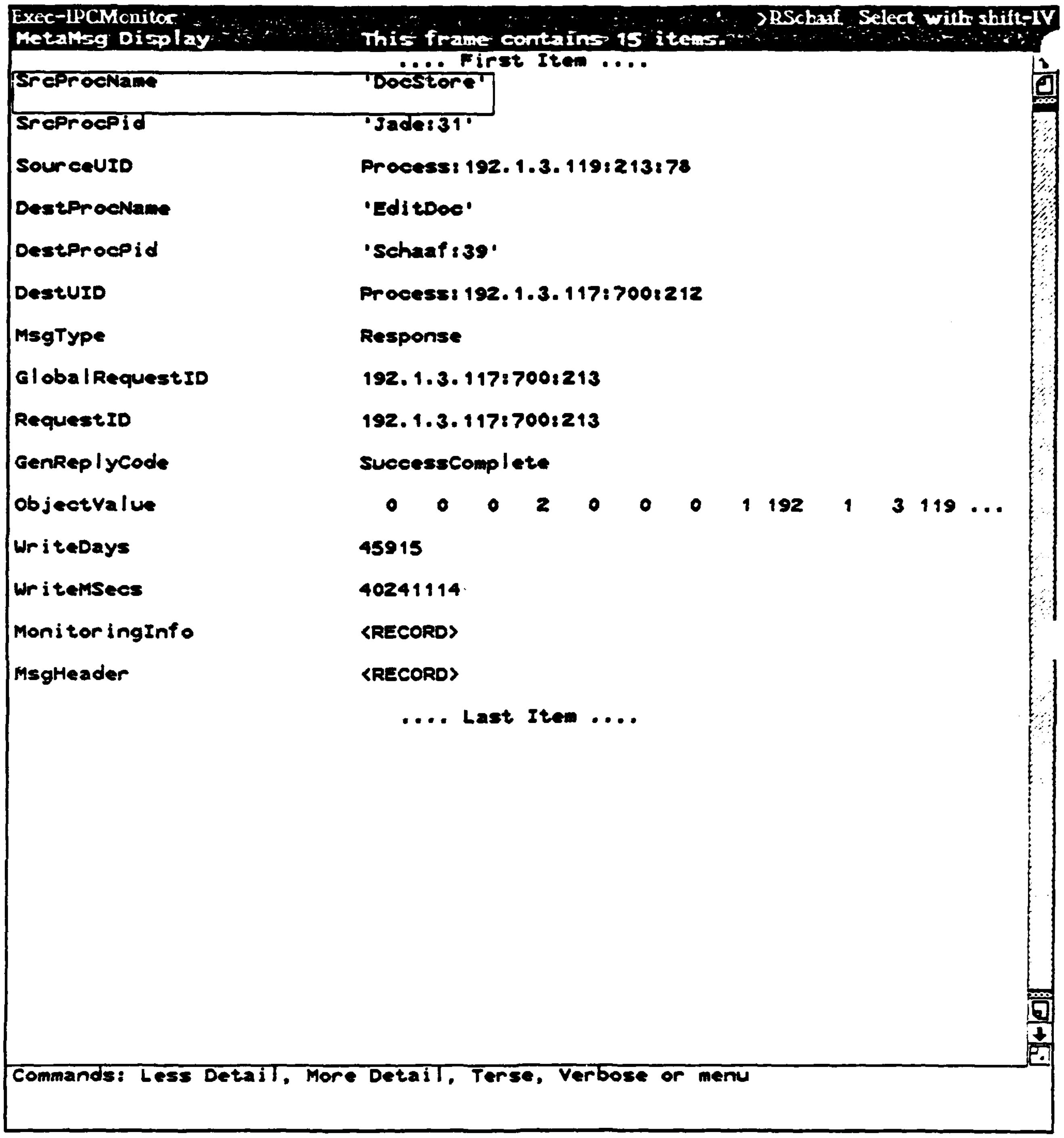
Figure 13. AuthBindingOf response



```
Source UID = Process:192.1.3.119:213:78
Dest UID = Process:192.1.3.117:700:212
Global Request ID = 192.1.3.117:700:213
Request ID = 192.1.3.117:700:213
General Reply Code = SuccessComplete
```

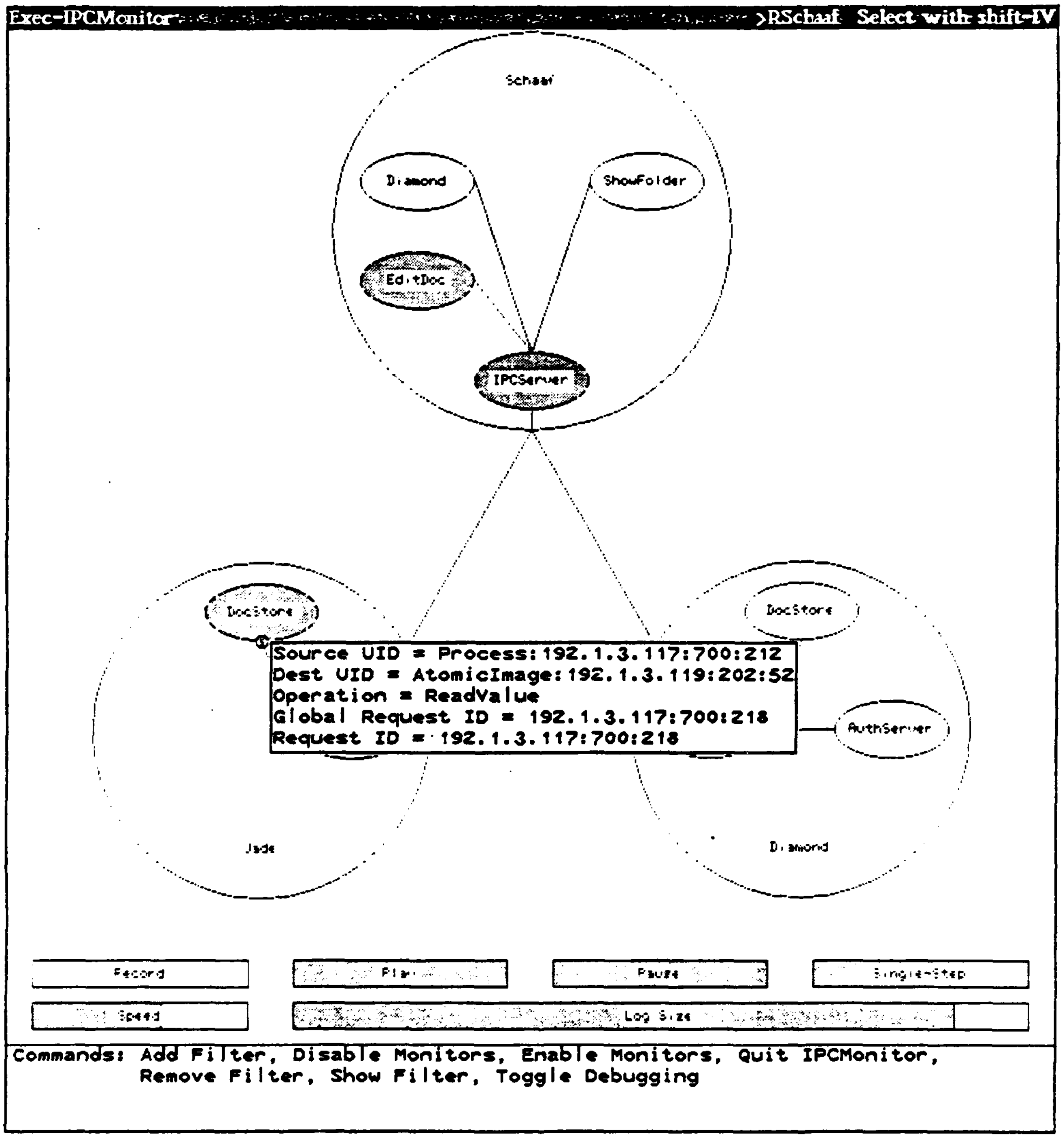
```
Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
          Remove Filter, Show Filter, Toggle Debugging
```

Figure 14. DocStruc ReadValue response



```
Exec-IPCMonitor                                    >RSchaaf  Select with shift-IV
MetaMsg Display          This frame contains 15 items.
                          .... First Item ....
SrcProcName                 'DocStore'

SrcProcPid                  'Jade:31'

SourceUID                   Process:192.1.3.119:213:78

DestProcName                'EditDoc'

DestProcPid                 'Schaaf:39'

DestUID                     Process:192.1.3.117:700:212

MsgType                     Response

GlobalRequestID             192.1.3.117:700:213

RequestID                   192.1.3.117:700:213

GenReplyCode                SuccessComplete

ObjectValue                   0   0   0   2   0   0   0   1 192   1   3 119 ...

WriteDays                   45915

WriteMSecs                  40241114

MonitoringInfo              <RECORD>

MsgHeader                   <RECORD>

                          .... Last Item ....

Commands: Less Detail, More Detail, Terse, Verbose or menu
```

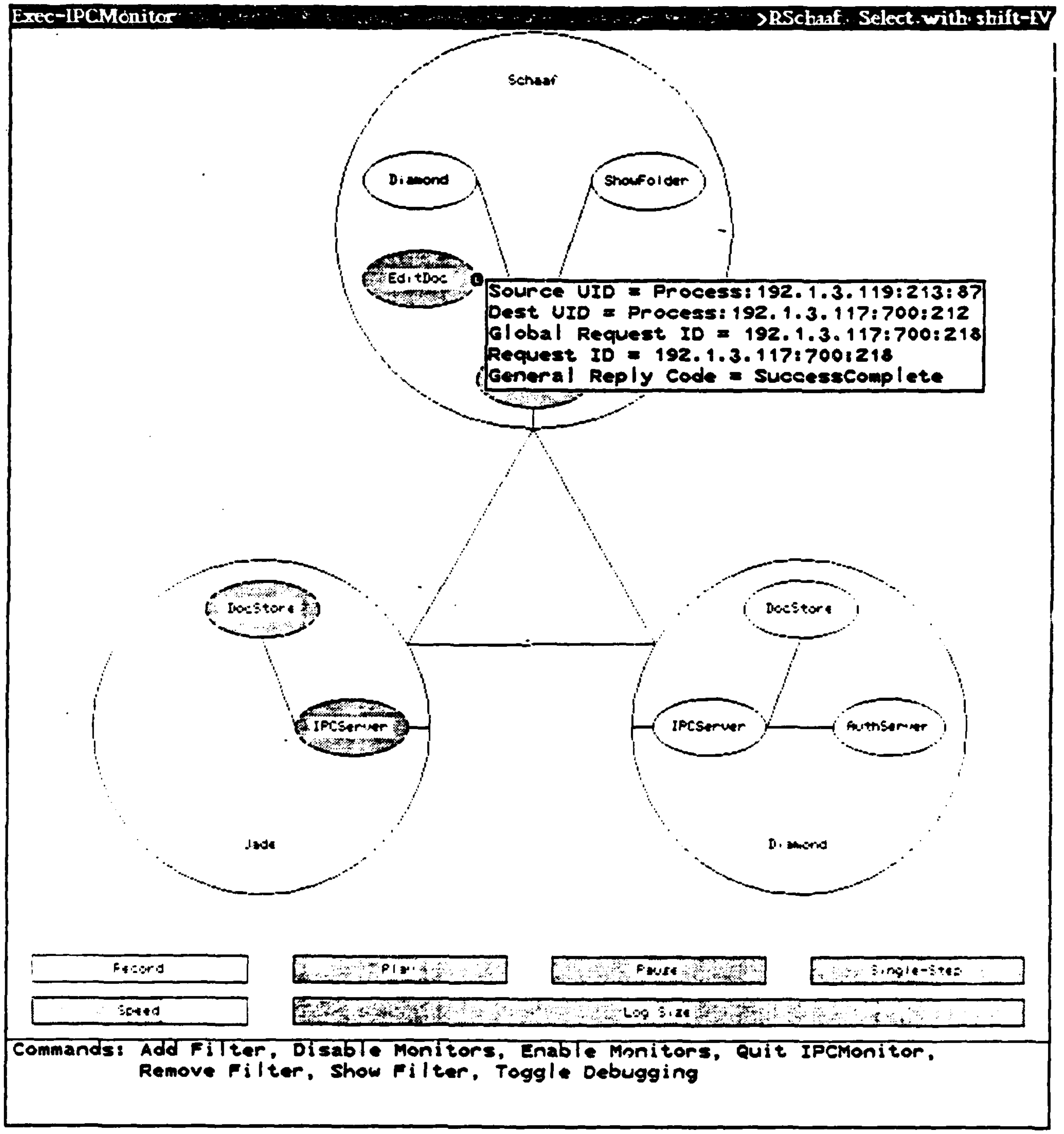
Figure 15. DocStruc ReadValue response - detailed information



```
Source UID = Process:192.1.3.117:700:212
Dest UID = AtomicImage:192.1.3.119:202:52
Operation = ReadValue
Global Request ID = 192.1.3.117:700:218
Request ID = 192.1.3.117:700:218
```

```
Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
          Remove Filter, Show Filter, Toggle Debugging
```

Figure 16. AtomicImage ReadValue request



```
Source UID = Process:192.1.3.119:213:87
Dest UID = Process:192.1.3.117:700:212
Global Request ID = 192.1.3.117:700:218
Request ID = 192.1.3.117:700:218
General Reply Code = SuccessComplete
```

```
Commands: Add Filter, Disable Monitors, Enable Monitors, Quit IPCMonitor,
          Remove Filter, Show Filter, Toggle Debugging
```

Figure 17. AtomicImage ReadValue response

7. JERICHO INTERLISP

The objective of the Jericho Interlisp task is to port and extend the Interlisp programming language and environment to BBN's Jericho personal computer. Interlisp is one of the two major dialects of the LISP programming language which underlies most research in Artificial Intelligence. This task services two principal goals. First, it provides the development environment in which other DARPA supported research in Artificial Intelligence proceeds at BBN, most notably the work in natural language understanding and knowledge representation. Second, this task builds the foundation for the ALEPH component of this project which is to explore novel programming techniques and tools.

The task of porting Interlisp-10 from a mainframe to a personal computer involves three categories of effort: 1) porting the initial system, 2) extending it to accommodate the functional capabilities of the hardware, and 3) system maintenance. In previous reporting periods, we have addressed each of these areas. In the current period, we incorporated Active Values, and we incorporated a multiple-process capability.

7.1 Active Values

We have implemented a version of Active Values which is modeled on the design used by Bobrow and Stefik in LOOPS. Active Values provide a way to invoke a function when the value of a variable (more specifically, a SPECVAR) is set, or when a field in a userdatatype, array record, block record, or array is either fetched or replaced. With them it is possible to implement monitoring (i.e.: "Call me if someone attempts to set my value to XYZ") and other event dependent mechanisms.

In our implementation, an Active Value is a datatype containing a local state, a PUT function, and a GET function. The local state holds the "real value", which may be another active value so Active Values can be nested. The PUT function is invoked whenever the Active Value is set or replaced. The GET function is invoked whenever the Active Value is fetched, except for SPECVARs where it is ignored.

PUT functions are invoked with arguments DATUM, OFFSET, ACTVAL, and NEWVAL. OFFSET is ignored for SPECVARS, ACTVAL is the active value containing the PUT function, and NEWVAL is the value to be stored. The usual form of a PUT function is:

```
    (LAMBDA (DATUM OFFSET ACTVAL NEWVAL)
         (DECLARE (LOCALVARS . T))
         (* User's code here)
         (PUTLOCALSTATE ACTVAL NEWVAL DATUM OFFSET)
```

PUTLOCALSTATE stores NEWVAL in the local state of ACTVAL which may invoke additional PUT functionS, with NEWVAL ending up in the local state of the innermost active value.

The arguments to a GET function are DATUM, OFFSET and ACTVAL. The usual form of a GET function is

```
    (LAMBDA (DATUM OFFSET ACTVAL)
         (DECLARE (LOCALVARS. T))
         (PROG1 (GETLOCALSTATE ACTVAL DATUM OFFSET)
                (* User's code)
```

GETLOCALSTATE gets the localstate of ACTVAL which may invoke additional GET funrtions. The reason for doing the GETLOCALSTATE first is to mimic the Loops nesting in which nested PUT functions are invoked from outermost to innermost while GET functions are invoked innermost first.

It is essential that the ACTVAL argument to a PUT function or GET function be a <u>LOCALVAR</u>. Therefore, PUT functions and GET functions must be <u>compiled</u>. Binding a SPECVAR to an Active Value will access the local state, not the Active Value.

A number of functions have been provided for the benefit of users.

To aid in defining PUT and GET functions, a function named DEFAVFN (NAME PUTFLG) has been provided. If PUTFLG is T, the function provides an editor template for defining NAME as a PUT function. If PUTFLG is NIL, the template provided is for a GET function.

To access the local state of an Active Value without triggering nested Active Values the functions are:

```
(GETLOCALSTATEONLY ACTVAL)
(PUTLOCALSTATEONLY ACTVAL NEWVAL)
```

To create Active Values, the function is:

```
(MAKEACTIVEVAR VAR PUTFN  VAL)
```

which makes the current binding of VAR an Active Value. PUTFN will be invoked when the current binding of VAR is set. If VAL is not NIL it becomes the localstate. Otherwise the local state is the current value of VAR. (Note that PUTFN is invoked when the current binding is set, not when VAR is rebound.)

The function

```
(MAKEACTIVEFIELD DATUM FIELDNAME PUTFN GETFN VAL)
```

makes the field FIELDNAME in DATUM active. VAL defaults as in MAKEACTIVEVAR. FIELDNAME is the record field name as in *fetch* or *replace*. FIELDNAME can also be an integer to allow activating unnamed array elements.

Other useful functions are:

```
(BREAKVAR VAR)
```

which is similar to BREAK(FN) in that a break is entered whenever the current binding of VAR is set;

```
(UNBREAKVAR VAR)
```

which removes the break;

```
(BREAKFIELD DATUM FIELDNAME WHEN)
```

which causes a break when the field FIELDNAME in DATUM is referenced;

```
(GETAVFNS DATUM FIELDNAME)
```

which aids in looking at Active Values by producing a list of GET function, PUT function pairs, outermost first;

```
(UNBREAKFIELD DATUM FIELDNAME)
```

which removes the break; and

```
(REMOVEACTIVEVALUE DATUM FIELDNAME PUTFN GETFN)
```

which removes an active value, anywhere in the nesting, from field FIELDNAME in DATUM. PUTFN and GETFN are usually LITATOMS. The Active Value removed is the first one found whose PUTFN adn GETFN match. If only one of PUTFN and GETFN is specified and an Active Value is found which has a function matching the one specified that function is set to NIL in the Active Value.

If DATUM is a LITATOM, FIELDNAME is ignored. To remove an active value directly PUTFN can be an Active Value. However, it must be remembered that passing an Active Value around requires that it not be bound to a SPECVAR, therefore not to any variable in interpreted code.

8. ALEPH

The goal of the ALEPH component of the project is to conceive and test new ideas and tools which can aid the programmer in his task. We expect that the capabilities of a personal computer such as the Jericho can offer new opportunities in this area, particularly the high-resolution bitmapped display.

In this reporting period, we improved the Graphical Debugger and the Code Presenter that were documented in previous reports, and we provided an application of Active Values that enables users to visualize the changes that have occurred between successive versions of software.

8.1 Application of Active Values

With the emergence of Active Values in INTERLISP Jericho we have begun to explore their use in helping software developers visualize changes in the data structures they manipulate. Our paradigm is as follows: a user looks at a screen in which figural representations of the different data structures he/she is working with are displayed. As the user's program modifies the data structure, he/she notices a change in the visual appearance of one of the representations. From this change, the user is able to infer effortlessly what went on, and modify his program accordingly. Active Values, i.e.: the ability to invoke any function when a value is either read or written on, make possible the cost-effective realization of this paradigm.

One of the visualization ideas we explored is to represent a list structure as a tree. For example, the list (A (B C) D) could be represented as

```
              B —— C
             /
A —— [] —— D
```

After implementing a set of functions to construct these trees, we experimented with various ways of visually portraying changes of different types. As none of these portrayals seemed satisfactory, we decided to try a different approach. Upon detection of a change, we would invoke a comparator that, given the original list and the one with a changes in it, would induce the procedural information needed to transform the original list into the new one.

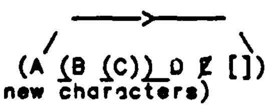
To be really useful, this procedural information must be represented in such a way that:

1. it corresponds closely to the annotations and scribbles a human editor would use in order to visually convey the changes he wants,
2. it must implicitly contain the precise instructions needed by a machine editor to carry out those changes.

Thus, comparing the lists

```
(A B C D E)    and    ((B (C)) D A)
```

we want to come up with a representation that enables us to portray the changes visually as, for instance,



(underlines denote new characters)

and capable of generating the sequence of INTERLISP editor commands,

```
*bi 3
*bi 2 3
*(4)
*move 1 to a 3
```

The representation we have chosen is based on the use of annotations which are affixed like labels to the original list's elements. The labels are MOVED-FROM, MOVED-TO, NEW, SAME, and BLANK. For the original list given in the example above, the representation would be:

```
((A . MOVED-FROM)
 (LP .  NEW)
   (B . SAME)
   (LP . NEW)
     (C . SAME)
   (RP . NEW))
 (RP . NEW))
 (D .  SAME)
 (E . BLANK)
 (A . MOVED-TO))
```

where for clarity we have used LP and RP instead of "(" and ")".

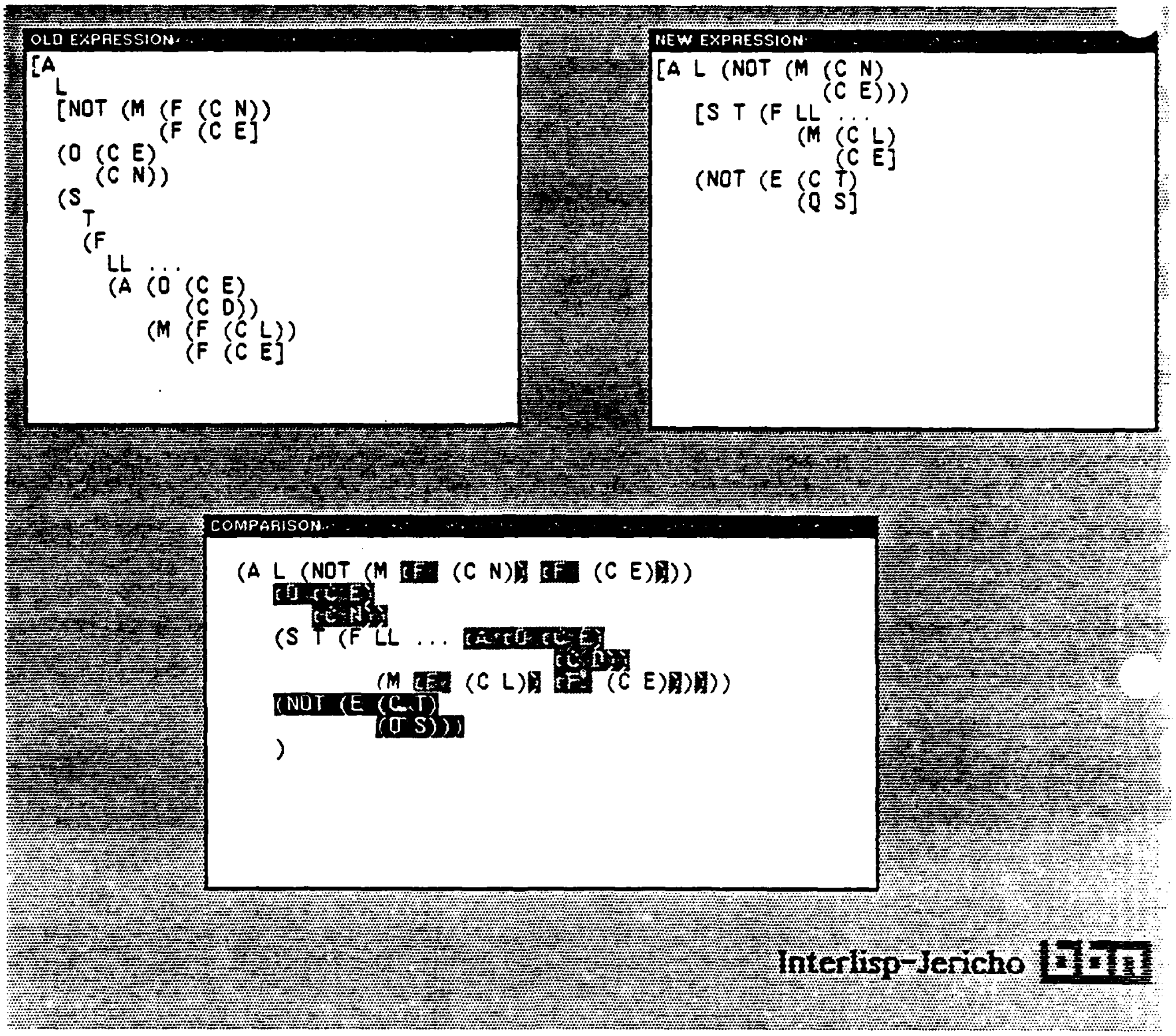
Notice how different this representation is from something like

```
((A . BLANK)
 ((B (C)) . NEW)
 (B . BLANK)
 (C . BLANK)
 (D . SAME)
 (E . BLANK)
 (A . NEW))
```

The former recognizes that -- B C -- can be transformed into -- (B (C)) -- by inserting parentheses, and that A is moved from being first in the list to being last. It corresponds to the "penciled" marks a human editor would make to convey how to transform one list into the other in a sensible way. The latter does the job but in a blind, brute force way: A is deleted from the beginning and inserted at the end as if it had never been seen before, and (B (C)) is considered a new element while B and C are deleted.

Once the representation has been induced, there are at least two ways of utilizing it. The first one is to obtain the corresponding commands for the INTERLISP structural editor. The software to accomplish this was designed but was not implemented due to lack of resources. The second, and more interesting one, is to generate from it something akin to the annotations used by human editors to convey visually to a typist the nature and extent of the changes they want.

We have produced a package that performs a version of this visual rendition. An example of what it can do is given in Figure 18.



```
[A
  L
  [NOT (M (F (C N))
          (F (C E]
  (O (C E)
     (C N))
  (S
    T
    (F
      LL ...
      (A (O (C E)
            (C D))
         (M (F (C L))
            (F (C E]
```

```
[A L (NOT (M (C N)
             (C E)))
   [S T (F LL ...
           (M (C L)
              (C E]
   (NOT (E (C T)
           (Q S]
```

```
(A L (NOT (M (F (C N)) (F (C E))))
   (O (C E)
      (C N))
   (S T (F LL ... (A (O (C E)
                        (C D))
           (M (F (C L)) (F (C E))))))
   (NOT (E (C T)
           (Q S)))
   )
```

Figure 18. Transforming Old into New

The window titled "NEW EXPRESSION" contains the result of several editing changes performed on the list contained in "NEW EXPRESSION". The window titled "COMPARISON" contains an annotated version of OLD EXPRESSION, where white characters on gray shade represent deletions, and inverted characters (white on black) represent new insertions.

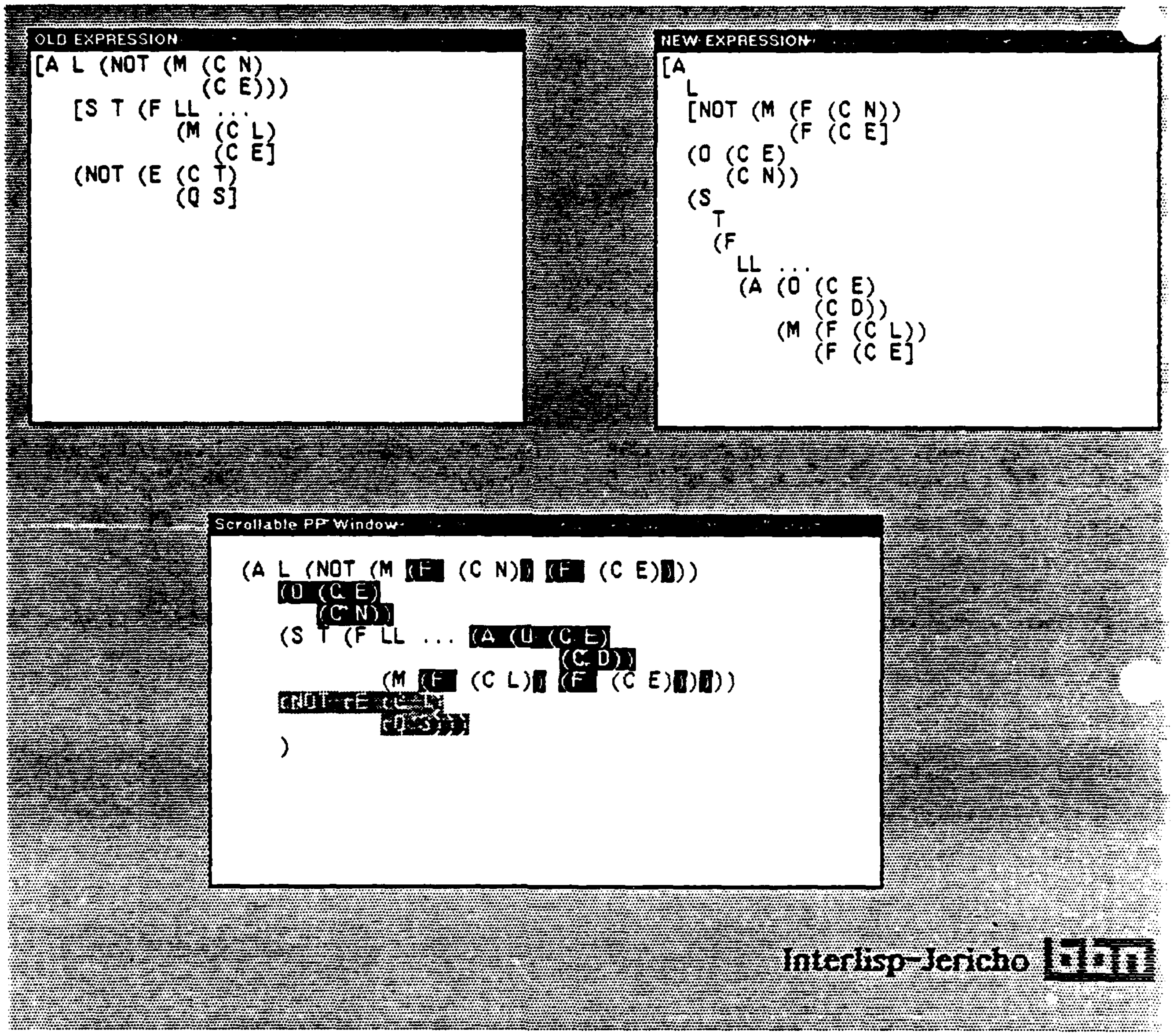
In Figure 18 we show the result of reversing the comparison by swapping "new" and "old".

8.2 Code Presenter

We have continued developing the code presenter reported in the previous report. The code presenter is a tool that helps a programmer examine code that usually has a high branching factor. Typically, at a given time, the programmer is interested in a single branch or a set of related branches. The code presenter provides two ways of looking at code that draw attention to the relevant paths in the code. The code presenter looks at variable bindings in a hypothetical environment and possibly in the running environment and presents the code with branches that are relevant to the environment in a visually distinctive way.

In the previous version of the code presenter, the code was compressed so that the irrelevant branches were eliminated and the relevant branches were partially evaluated. The resulting code is functionally equivalent to the original code in the given environment. In the current version, we have extended the code presenter to give the programmer the option of seeing the compressed code or seeing the original code with the relevant branches highlighted.

In a given environment, a piece of code will either be evaluated, it will not be evaluated, or there is not enough information in the environment to know whether it will be evaluated. The last class of code is treated as two classes, code that contains unbound variables that prevent further analysis, and code that is not available to be analyzed (usually compiled code whose source definitions are not available). The details of the analysis were given in the previous report. It then presents the code using four different fonts to highlight the branches. Code that will be executed in the given environment is highlighted in a bold font. Code that may be executed depending on the values of bound variables is presented in a normal font. Code that can not be analyzed is presented in an italic font. And code that will not be evaluated in the



```
[A L (NOT (M (C N)
             (C E)))
   [S T (F LL ...
           (M (C L)
              (C E]
   (NOT (E (C T)
           (Q S]
```

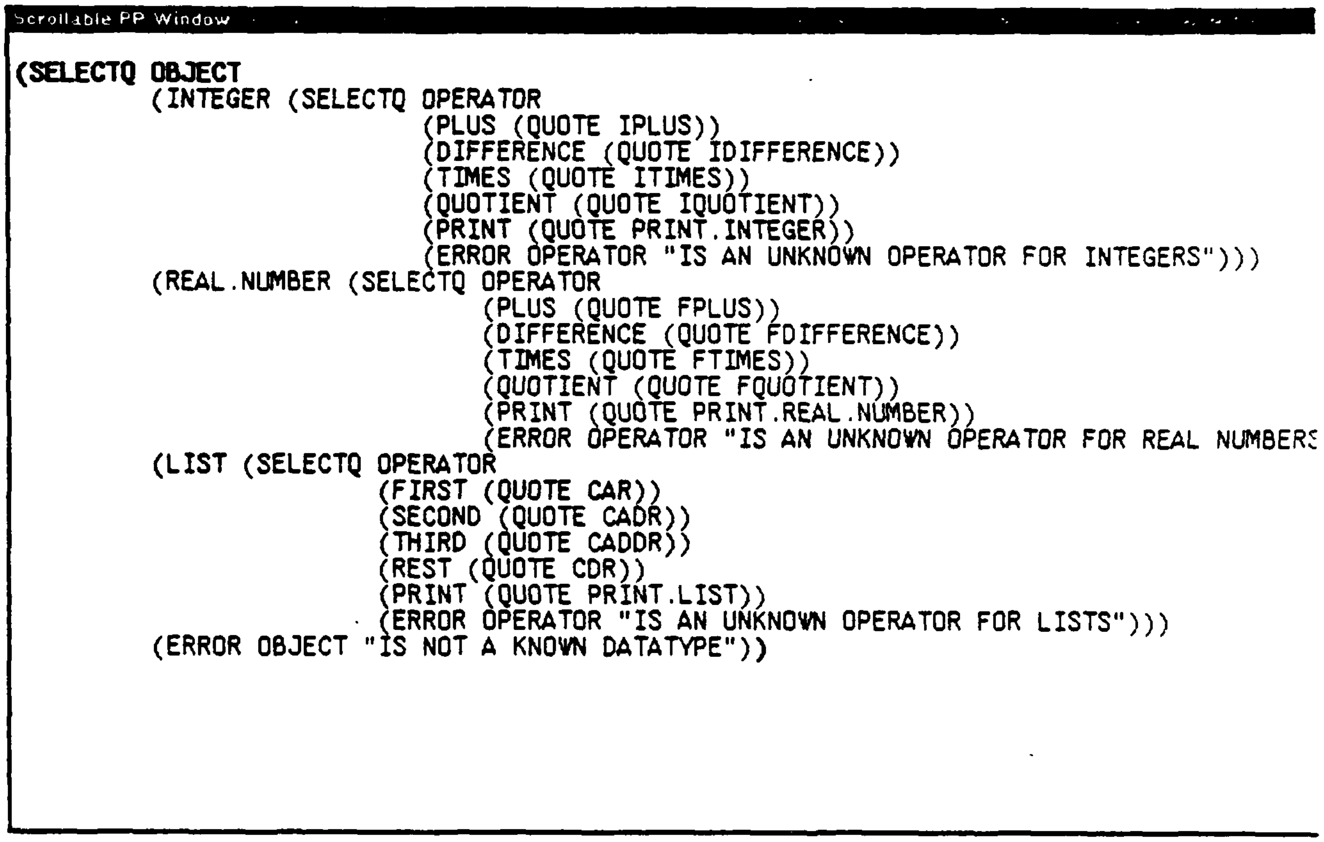
```
[A
 L
 [NOT (M (F (C N))
         (F (C E]
 (O (C E)
    (C N))
 (S
  T
  (F
   LL ...
   (A (O (C E)
         (C D))
      (M (F (C L))
         (F (C E]
```

```
(A L (NOT (M (F (C N)) (F (C E))))
   (O (C E)
      (C N))
   (S T (F LL ... (A (O (C E)
                        (C D))
               (M (F (C L)) (F (C E)))))
   (NOT (E (C T)
           (Q S)))
   )
```

Figure 19. The reverse transformation

current environment is presented in a small *font*.

Figure 20 is an example of a piece of code with a fairly high branching factor. All examples in this section use the same code and environments as the examples in



```
(SELECTQ OBJECT
        (INTEGER (SELECTQ OPERATOR
                          (PLUS (QUOTE IPLUS))
                          (DIFFERENCE (QUOTE IDIFFERENCE))
                          (TIMES (QUOTE ITIMES))
                          (QUOTIENT (QUOTE IQUOTIENT))
                          (PRINT (QUOTE PRINT.INTEGER))
                          (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR INTEGERS")))
        (REAL.NUMBER (SELECTQ OPERATOR
                              (PLUS (QUOTE FPLUS))
                              (DIFFERENCE (QUOTE FDIFFERENCE))
                              (TIMES (QUOTE FTIMES))
                              (QUOTIENT (QUOTE FQUOTIENT))
                              (PRINT (QUOTE PRINT.REAL.NUMBER))
                              (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR REAL NUMBERS
        (LIST (SELECTQ OPERATOR
                       (FIRST (QUOTE CAR))
                       (SECOND (QUOTE CADR))
                       (THIRD (QUOTE CADDR))
                       (REST (QUOTE CDR))
                       (PRINT (QUOTE PRINT.LIST))
                       (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR LISTS")))
        (ERROR OBJECT "IS NOT A KNOWN DATATYPE"))
```

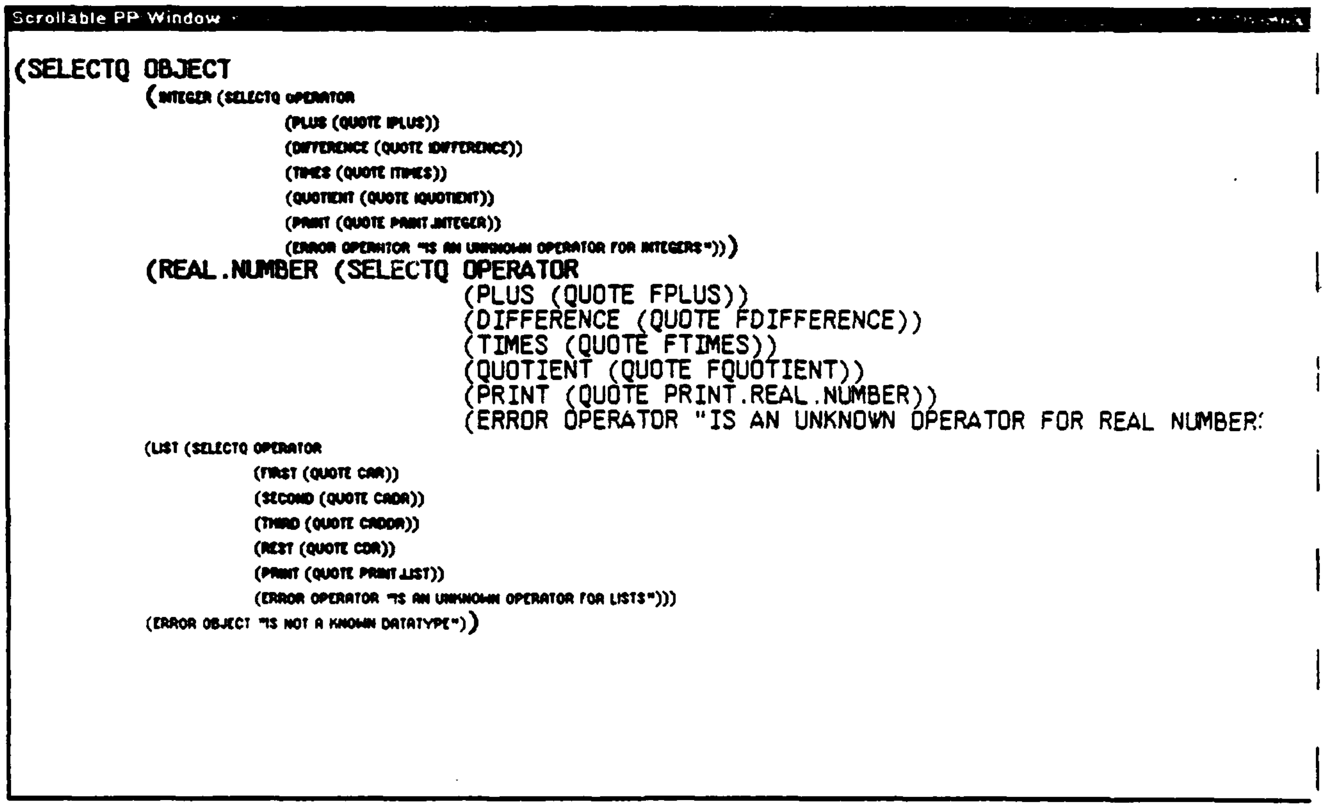
Figure 20. Code segment parameterized by variables OBJECT and OPERATOR

the previous report.

The code in figure 20 is parameterized by the variables OBJECT and OPERATOR and is intended to return the name of the function which performs the prescribed operation for the given object type. For example, the PLUS operator for the INTEGER object is IPLUS. Since the figure shows the code segment displayed when both variables OBJECT and OPERATOR have no value, the entire code segment is printed in the normal font.

Figure 21 shows what happens when the variable OBJECT is bound to

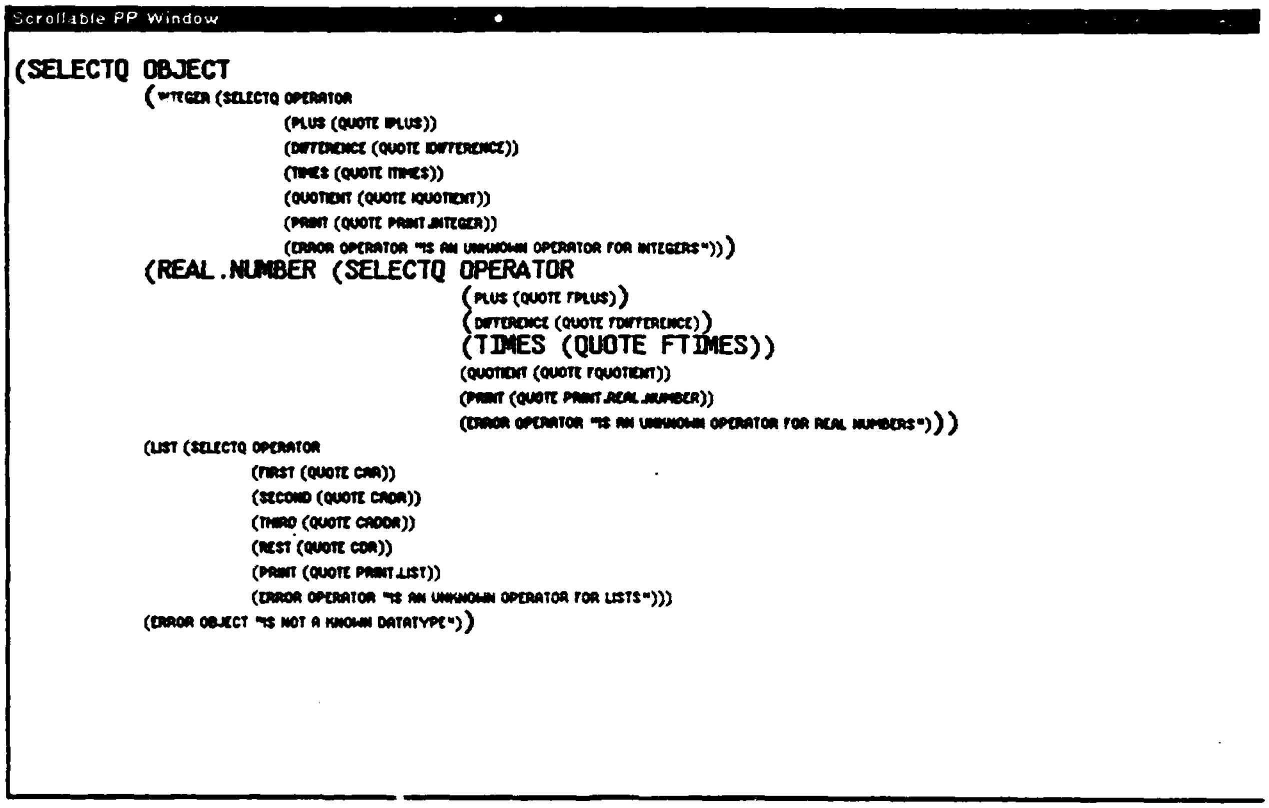
REAL.NUMBER and OPERATOR is unbound. Notice that the outer SELECTQ is in bold face to signify that it is evaluated, and so are the parentheses around the INTEGER case. The contents of the list, however, are in a small font to signify that the INTEGER case will be investigated but will fail because INTEGER is not the same as REAL.NUMBER. The REAL.NUMBER case is in bold face as is the SELECTQ it contains. The cases of the inner SELECTQ are in the normal font because the variable OPERATOR is not bound. The code presenter can not determine which of the cases will execute in an environment with OPERATOR unbound.



```
(SELECTQ OBJECT
         (INTEGER (SELECTQ OPERATOR
                           (PLUS (QUOTE IPLUS))
                           (DIFFERENCE (QUOTE IDIFFERENCE))
                           (TIMES (QUOTE ITIMES))
                           (QUOTIENT (QUOTE IQUOTIENT))
                           (PRINT (QUOTE PRINT.INTEGER))
                           (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR INTEGERS")))
         (REAL.NUMBER (SELECTQ OPERATOR
                               (PLUS (QUOTE FPLUS))
                               (DIFFERENCE (QUOTE FDIFFERENCE))
                               (TIMES (QUOTE FTIMES))
                               (QUOTIENT (QUOTE FQUOTIENT))
                               (PRINT (QUOTE PRINT.REAL.NUMBER))
                               (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR REAL NUMBER
         (LIST (SELECTQ OPERATOR
                        (FIRST (QUOTE CAR))
                        (SECOND (QUOTE CADR))
                        (THIRD (QUOTE CADDR))
                        (REST (QUOTE CDR))
                        (PRINT (QUOTE PRINT.LIST))
                        (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR LISTS")))
         (ERROR OBJECT "IS NOT A KNOWN DATATYPE"))
```

Figure 21. Code segment with OBJECT bound to REAL.NUMBER and OPERATOR unbound

Figure 22 shows the code presented when OBJECT is bound to REAL.NUMBER and OPERATOR is bound to TIMES. It shows that the branch to be taken is fully determined and that the value returned will be FTIMES.

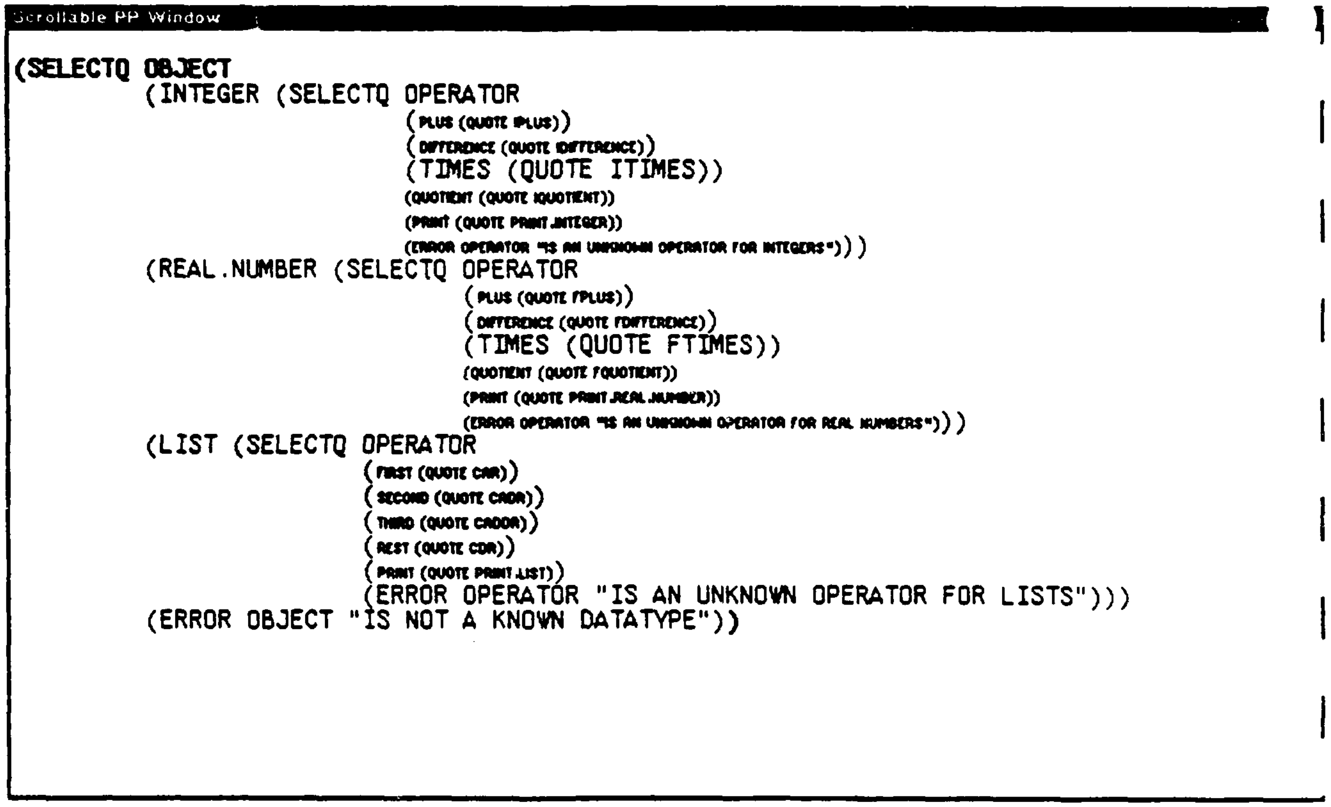


```
(SELECTQ OBJECT
          (INTEGER (SELECTQ OPERATOR
                            (PLUS (QUOTE IPLUS))
                            (DIFFERENCE (QUOTE IDIFFERENCE))
                            (TIMES (QUOTE ITIMES))
                            (QUOTIENT (QUOTE IQUOTIENT))
                            (PRINT (QUOTE PRINT.INTEGER))
                            (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR INTEGERS")))
          (REAL.NUMBER (SELECTQ OPERATOR
                                (PLUS (QUOTE FPLUS))
                                (DIFFERENCE (QUOTE FDIFFERENCE))
                                (TIMES (QUOTE FTIMES))
                                (QUOTIENT (QUOTE FQUOTIENT))
                                (PRINT (QUOTE PRINT.REAL.NUMBER))
                                (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR REAL NUMBERS")))
          (LIST (SELECTQ OPERATOR
                         (FIRST (QUOTE CAR))
                         (SECOND (QUOTE CADR))
                         (THIRD (QUOTE CADDR))
                         (REST (QUOTE CDR))
                         (PRINT (QUOTE PRINT.LIST))
                         (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR LISTS")))
          (ERROR OBJECT "IS NOT A KNOWN DATATYPE"))
```

Figure 22. Code segment with OBJECT bound to REAL.NUMBER and OPERATOR bound to TIMES

Figure 23 shows the code presented when OBJECT is unbound and OPERATOR is bound to TIMES. Notice that the path in each of the inner selectqs is fully determined. The outer selectq can not determine which branch to take in the given environment. In this example, one of four values will be returned, ITIMES, FTIMES, or one of the two error statements.

The Code Presenter is currently running in Jericho INTERLISP and it can handle most INTERLISP functions including display oriented functions. It has proven to be a



```
(SELECTQ OBJECT
         (INTEGER (SELECTQ OPERATOR
                           (PLUS (QUOTE IPLUS))
                           (DIFFERENCE (QUOTE IDIFFERENCE))
                           (TIMES (QUOTE ITIMES))
                           (QUOTIENT (QUOTE IQUOTIENT))
                           (PRINT (QUOTE PRINT.INTEGER))
                           (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR INTEGERS")) )
         (REAL.NUMBER (SELECTQ OPERATOR
                               (PLUS (QUOTE FPLUS))
                               (DIFFERENCE (QUOTE FDIFFERENCE))
                               (TIMES (QUOTE FTIMES))
                               (QUOTIENT (QUOTE FQUOTIENT))
                               (PRINT (QUOTE PRINT.REAL.NUMBER))
                               (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR REAL NUMBERS")) )
         (LIST (SELECTQ OPERATOR
                        (FIRST (QUOTE CAR))
                        (SECOND (QUOTE CADR))
                        (THIRD (QUOTE CADDR))
                        (REST (QUOTE CDR))
                        (PRINT (QUOTE PRINT.LIST))
                        (ERROR OPERATOR "IS AN UNKNOWN OPERATOR FOR LISTS")))
         (ERROR OBJECT "IS NOT A KNOWN DATATYPE"))
```

Figure 23. Code segment with OBJECT unbound and OPERATOR bound to TIMES

useful tool for analyzing and debugging complicated functions that the user does not fully understand or that have a non-trivial logical bug.

One limitation of the Code Presenter is that it is not a passive tool -- it requires that the programmer set up environments and select code to examine. Both are trivial to do but the programmer must examine the code and decide what variables to bind in the environment.

Another limitation is that neither method of display (highlighting or compressing) is really enough to do the entire job alone. Highlighting of a large complicated function can be hard to read. And the compression of a large complicated function does not resemble the original code. Clearly the ideal solution is some combination of the two methods. Unfortunately we were not able to try to combine them. It would require a great deal of research to determine a useful balance between the two. The balance is a function of the complexity of the code, the size of the code, the restrictions implied by the variable bindings in the environment, and the programmers ability to comprehend the size and complexity of the code presented.

A less important limitation we encountered is that the INTERLISP printing package was not designed to handle fonts properly. We were able to obtain the output shown here only by improving the PRETTYPRINT package but it is obviously not perfect. It does not handle line height properly when fonts are of different heights. Printing characters in different fonts can cause overlapping characters both vertically and horizontally.

9. HERMES MAINTENANCE

During this period work on the Hermes electronic mail system program was routine maintenance.

DISTRIBUTION LIST

addresses	number of copies
Thomas F. Lawrence RADC/COTC	10
RADC/COVL GRIFFISS AFB NY 13441	1
RADC/DAP GRIFFISS AFB NY 13441	2
ADMINISTRATOR DEF TECH INF CTR ATTN: DTIC-DDA CAMERON STA BG 5 ALEXANDRIA VA 22304-6145	5
RADC/COTD BLDG 3, ROOM 16 GRIFFISS AFB NY 13441-5700	1
AFCSA/SAMI Attn: Miss Griffin 10363 Pentagon Wash DC 20330-5425	1
HQ USAF/SCTT Pentagon Wash DC 20330-5190	1
SAF/AQSC Pentagon 4D-267 Wash DC 20330-1000	1
DIRECTOR DMAHTC ATTN: SDSIM Wash DC 20315-0030	1

Director, Info Systems OASD (C3I) Rm 3E187 Pentagon Wash DC 20301-3040	1
Fleet Analysis Center Attn: GIDEP Operations Center Code 30G1 (E. Richards) Corona CA 91720	1
HQ AFSC/DLAE ANDREWS AFB DC 20334-5000	1
HQ AFSC/XRT Andrews AFB MD 20334-5000	1
HQ AFSC/XRK ANDREWS AFB MD 20334-500	1
HQ SAC/SCPT OFFUTT AFB NE 68113-5001	1
HQ ESC/DOQR Attn: Fred Ladwig San Antonio TX 78243-5000	1
DTESA/RQEE ATTN: LARRY G. MCMANUS 2501 YALE STREET SE Airport Plaza, Suite 102 ALBUQUERQUE NM 87106	1
HQ TAC/DRIY Attn: Mr. Westerman Langley AFB VA 23665-5001	1

HQ TAC/DCA LANGLEY AFB VA 23665-5001	1
HQ TAC/DRCC LANGLEY AFB VA 23665-5001	1
HQ TAC/DRCA LANGLEY AFB VA 23665-5001	1
ASD/ENEMS Wright-Patterson AFB OH 45433-6503	2
ASD-AFALC/AXP WRIGHT-PATTERSON AFB OH 45433	1
ASD/AFALC/AXAE Attn: W. H. Dungey Wright-Patterson AFB Oh 45433-6533	1
ASD/ENAMA Wright-Patterson AFB OH 45433	1
AFIT/LDEE BUILDING 640, AREA B WRIGHT-PATTERSON AFB OH 45433-6583	1
AFWAL/MLPC WRIGHT-PATTERSON AFB OH 45433-6533	1
AFWAL/MLTE WRIGHT-PATTERSON AFB OH 45433	1

AFWAL/FIES/SURVIAC WRIGHT-PATTERSON AFB OH 45433	1
AAMRL/HE WRIGHT-PATTERSON AFB OH 45433-6573	1
Air Force Human Resources Laboratory Technical Documents Center AFHRL/LRS-TDC Wright-Patterson AFB OH 45433	1
2750 ABW/SSLT Bldg 262 Post 11S Wright-Patterson AFB OH 454433	1
AFHRL/OTS WILLIAMS AFB AZ 85240-6457	1
1843EIG/EIEM HICKAM AFB HI 96854	1
AUL/LSE MAXWELL AFB AL 36112-5564	1
HQ AFSPACECOM/XPYS ATTN: DR. WILLIAM R. MATOUSH PETERSON AFB CO 80914-5001	1
3280TTG/EISS Attn: TSgt Kirk Lackland AFB TX 78236	1

Defense Communications Engineering Ctr Technical Library 1860 Wiehle Avenue Reston VA 22090-5500	1
COMMAND CONTROL AND COMMUNICATIONS DIV DEVELOPMENT CENTER MARINE CORPS DEVELOPMENT & EDUCATION COMMAND ATTN: CODE DICA QUANTICO VA 22134-5080	2
AFLMC/LGY ATTN: CH, SYS ENGR DIV GUNTER AFS AL 36114	1
U.S. Army Strategic Defense Command Attn: DASD-H-MPL P.O. Box 1500 Huntsville AL 35807-3801	1
COMMANDING OFFICER NAVAL AVIONICS CENTER LIBRARY - D/765 INDIANAPOLIS IN 46219-2189	1
COMMANDING OFFICER NAVAL TRAINING SYSTEMS CENTER TECHNICAL INFORMATION CENTER BUILDING 2068 ORLANDO FL 32813-7100	1
COMMANDER NAVAL OCEAN SYSTEMS CENTER ATTN: TECHNICAL LIBRARY, CODE 96428 SAN DIEGO CA 92152-5000	1
COMMANDER (CODE 3433) ATTN: TECHNICAL LIBRARY NAVAL WEAPONS CENTER CHINA LAKE, CALIFORNIA 93555-6001	1
SUPERINTENDENT (CODE 1424) NAVLA POST GRADUATE SCHOOL MONTEREY CA 93943-5000	1

COMMANDING OFFICER NAVAL RESEARCH LABORATORY ATTN: CODE 2627 WASHINGTON DC 20375-5000	2
SPACE & NAVAL WARFARE SYSTEMS COMMAND PMW 153-3DP ATTN: R. CAVARESE WASHINGTON DC 20363-5100	1
CDR, U.S. ARMY MISSILE COMMAND REDSTONE SCIENTIFIC INFORMATION CENTER ATTN: AMSMI-RD-CS-R (DOCUMENTS) REDSTONE ARSENAL AL 35898-5241	2
Advisory Group on Electron Devices Hammond John/Technical Info Coordinator 201 Varick Street, Suite 1140 New York NY 10014	2
UNIVERSITY OF CALIFORNIA/LOS ALAMOS NATIONAL LABORATORY ATTN: DAN BACA/REPORT LIBRARIAN P.O. BOX 1663, MS-P364 LOS ALAMOS NM 87545	1
RAND CORPORATION THE/LIBRARY HELFER DORIS S/HEAD TECH SVCS P.O. BOX 2138 SANTA MONICA CA 90406-2138	1
AEDC LIBRARY (TECH REPORTS FILE) MS-100 ARNOLD AFS TN 37389-9998	1
USASG Attn: ASH-PCA-CRT Ft Huachuca AZ 85613-6000	1
DOT LIBRARY/10A SECTION ATTN: M493.2 800 INDEPENDENCE AVE. S.W. WASH DC 20591	1

1839 EIG/EIET (KENNETH W. IREY) KEESLER AFB MS 39534-6348	1
JTFPMC Attn: Technical Director 1500 Planning Research Drive McLean VA 22102	2
HQ ESC/CWPP San Antonio TX 78243-5000	1
AFEWC/ESRI SAN ANTONIO TX 78243-5000	4
485 EIG/EIER (DMC) GRIFFISS AFB NY 13441-6348	2
ESD/AVS ATTN: ADV SYS DEV HANSCOM AFB MA 01731-5000	1
ESD/ICP HANSCOM AFB MA 01731-5000	1
ESD/AVSE BLDG 1704 HANSCOM AFB MA 01731-5000	2
HQ ESD SYS-2 HANSCOM AFB MA 01731-5000	1
ESD/TCD-2 ATTN: CAPTAIN J. MEYER HANSCOM AFB MA 01731-5000	1

The Scftware Ergineering Institute 1
Attn: Major Dan Burtcn, USAF
580 Scuth Aiker Avenue
Pittsburgh PA 15232-1502

DIRECTOR 1
NSA/CSS
ATTN: T513/TDL (DAVID MARJARUM)
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: W166
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: R-8316 (MR. ALLEY)
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: R24
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: R21
9800 SAVAGE ROAD
FORT GEORGE G MEASDE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: DEFSMAC (JAMES E. HILLMAN)
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: R31
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: R5
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: R8
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: SC31
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: S21
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: V33 (S. Friedrich)
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: WC7
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 1
NSA/CSS
ATTN: W3
FORT GEORGE G MEADE MD 20755-6000

DIRECTOR 2
NSA/CSS
ATTN: R523
FORT GEORGE G MEADE MD 20755-6000

DoD COMPUTER SECURITY CENTER 1
ATTN: C4/TIC
9800 SAVAGE ROAD
FORT GEORGE G MEADE MD 20755-6000

Harry C. Forsdick 1
BBN Laboratories Inc
10 Moulton Street
Cambridge, MA
02238

Jon Silverman 1
Honey Incorporated
Corporate Systems Development Center
1000 Boone Avenue (North)
Golden Valley, MN 55427-4437

James Richardson 1
Honeywell Incorporated
Corporate Systems Development Center
1000 Boone Avenue (North)
Golden Valley, MN 55427-4437

E. Douglas Jensen 1
Carnegie-Mellon University
Department of Computer Science
Schenley Park - Wean Hall
Pittsburg, PA 15213-3890

Joseph Lugo 1
Harris Corporation
Government Information Systems
Bldg. 1505, John Rodes Boulevard
Melbourne, FL 32935-9240

Peter M. Melliar-Smith 1
University of California (UCES)
Dept. of Electrical and Computer Engineering
Santa Barbara, CA
93106-0001

Teresa F. Lunt 1
SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, CA 94040-1234

Andrew Cromarty 1
Advanced Decisions Systems
Suite 236
201 San Antonio Circle
Mountain View, CA 94040-1234

Alan Lazarra 1
Knowledge System Concepts
262 Liberty Plaza
Rome, NY
13440-4460

Paul D. Stachour 1
Honeywell Incorporated
Secure Computing Technology Center
Suite 130, 2855 Anthony Lane (South)
St. Anthony, MN 55418-3265

Richard Schartz 1
Bolt Beranek and Newman
BBN Laboratories
10 Moulton Street
Cambridge, MA 02238-0001

Matthew Morgenstern 1
SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, CA 94025-3493

Henry Lefkovits 1
AOG
Harvard Depot Road
P. O. Box M
Harvard, MA 01451-0551

Richard LeBlank 1
Georgia Institute of Technology
School of Information and Computer Science
225 North Avenue
Atlanta, GA 30332-0280

Daniel A. Wheeler 1
PAR Government Systems Corporation
PAR Technology Park
220 Seneca Turnpike
New Hartford, NY 13413-1191

Hari Madduri 1
Honeywell Incorporated
Corporate Systems Division (MN63-C080)
1000 Boone Avenue (North)
Golden Valley, MN 55427-4437

Julian Center 1
Dynamics Research Corporation
60 Frontage Road
Andover, MA 01810-5414

Steve Vinter 1
Bolt Beranek and Newman
BBN Laboratories
10 Moulton Street
Cambridge, MA 02238-0001

Anita Skelton 1
Unisys Corporation
System Development Group
5151 Camino Ruiz
Camarillo, CA 93010-8601

Ron Peterson 1
Martin Marietta
Martin Marietta Aerospace
P. O. Box 179
Denver, CO 80201-0179

Deborah Teasdale 1
Naval Ocean Systems Center (NOSC)
Code 443,271
Catalina Boulevard
San Diego, CA 92152-5000

Andre VanTilborg 1
Office of Naval Research
Code 3311 - Room 704
800 North Quincy Street
Arlington, VA 22217-5000

Gerald Caprara 1
Kamar Sciences Corporation
258 Genesee Street
Utica, NY
13502-4636

J. Thomas Haigh 1
Honeywell Incorporated
SCTC (Suite 130)
2855 Anthony Lane (South)
St. Anthony, MN 55418-3265

Diane Smith 1
Computer Corporation of America
Four Cambridge Center
Cambridbe, MA
02142-1489

Mike Frankel 1
SRI International
Information Sciences and Technology Center
333 Ravenswood Avenue \
Menlo Park, CA 94025

Col. Alex Lancaster 1
Defense Advanced Research Planning Agency
DARPA/ISTC
1400 Wilson Boulevard
Arligton, VA 22209

John Cambell 1
National Security Agency
C31,9800 Savage Road
Ft. Meade, MD
20755-6000

Patricia Baskinger 1
IIT Research Institute
Turin Road (North)
P. O. Box 180
Rome, NY 13440-0180

Lt. Eric Bennett 1
Electronic Systems Division
ESD/MCN
Hanscom AFB, MA
01731-5000

Marvin Schaefer 1
Trusted Information Systems
P. O. Box 45
Glenwood, MD
21738

Richard C. White 1
Datalogic Systems Incorporated
Suite 209
3868 Carson Street
Torrance, CA 90503

Richard Platek 1
Odyssey Research Associates
1283 Trumansburg Road
Ithaca, NY
14850-1313

Bud Graf 1
United States Army
HQ CECOM
AMSEL-COM-IO
Ft. Monmouth, NJ 07703

Maj. Michael Smith 1
Joint Strategic Target Planning Staff
JSTPS/JKSF
Offutt AFB, NE
68113-5000

Robert Leary 1
Defense Communications Agency
DCA/C4S
1930 Isaac Newton Square
Reston, VA 22090-5003

Don Herkimer 1
Martin Marietta
Martin Marietta Aerospace (LC425)
P. O. Box 179
Denver, CO 80201-0179

Cathy Linn 1
Institute for Defense Analysis
1801 Beauregard Street
Alexandria, VA 22311-1772

Duane Northcutt 1
Carnegie-Mellon University
Department of Computer Science
Schenley Park - Wean Hall
Pittsburg, PA 15213-3890

Allan Seresey 1
United States Army
CECOM
AMSEL-RD-C3-IE-3
Ft. Monmouth, NJ 07703

William Smith 1
MITRE Corporation
Washington - W74,7525
Cold Shaire Drive
McLean, VA 22102

Louanna Notargiacomo 1
Unisys Corporation
7929 Westpark Drive
McLean, VA
22102

John M. Rushby 1
SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, CA 94025

Henry Bayard 1
MITRE Corporation
Network Technology Systems Department
P. O. Box 208
Bedford, MA 01730-0208

Terry V. C. Benzel 1
The MITRE Corporation
MS B330
Burlington Road
Bedford, MA 01730

Nicholas D. Murray 1
NASA
M/S 478
NASA Langley Research Center
Hampton, VA 23667

Steve Crocker 1
Trusted Information Systems
11340 West Olympic Boulevard
Suite 265
Los Angeles, CA 90064

Robert Sccui 1
ESD/ATS
Hanscom AFB, MA
01731-5000

Les Anderson 1
Naval Ocean Systems Center (NOSC)
Code 443,271
Catalina Boulevard
San Diego, CA 92152-5000

SDIO/P1-EN (LTC Sowa) 1
The Pentagon
Washington DC
20301-7100

SDIO/P1-BM (Capt Hart) 1
The Pentagon
Washington DC
20301-7100

SDIO/P1-EN (LTC Rindt) 1
The Pentagon
Washington DC
20301-7100

IDA (SDIO Library) 1
Albert Perella
1801 North Beauregard Street
Alexandria VA
22311

SAF/AGSD 1
LTC Ben Greenway
The Pentagon
Washington DC
20330

AFSC/CV-D 1
LTC Flynn
Andrews AFB, MD
20334-5000

HQ SD/XR 1
Col Heimach
P. O. Box 92960
Worldway Postal Center
Los Angeles, CA 90009-2960

HQ SD/CNC 1
Col Wilkerson
P. O. Box 92960
Worldway Postal Center
Los Angeles, CA 90009-2960

HQ SD/CNCI 1
Col Hohman
P. O. Box 92960
Worldway Postal Center
Los Angeles, CA 90009-2960

HQ SD/CNCIS 1
LTC Fennell
P. O. Box 92960
Worldway Postal Center
Los Angeles, CA 90009-2960

HQ SD/CNW 1
P. O. Box 92960
Worldway Postal Center
Los Angeles, CA 90009-2960

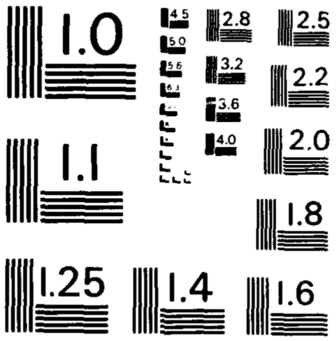
HQ SD/CWX 1
P. O. Box 92960
Worldway Postal Center
Los Angeles, Ca 90009-2960

HQ SD/CNP 1
P. O. Box 92960
Worldway Postal Center
Los Angeles, CA 90009-2960

ESD/AT 1
Col Paul
Hanscom AFB, MA
01731-5000

ESD/ATS 1
LTC Olderberg
Hanscom AFB, MA
01731-5000

AD-A203 780   RESEARCH IN DISTRIBUTED PERSONAL COMPUTER-BASED   2/2
INFORMATION SYSTEMS VOLUM..(U) BBN LABS INC CAMBRIDGE
MA H C FORSDICK ET AL. AUG 88 BBN-5901-VOL-1
UNCLASSIFIED RADC-TR-88-159-VOL-1 F30601-81-C-0256   F/G 12/7   NL


1.0
4.5
2.8
2.5
5.0
3.2
2.2
5.6
6.3
3.6
1.1
4.0
2.0
1.8
1.25
1.4
1.6

ESD/ATN 1
Col Leib
Hanscom AFB, MA
01713-5000

AFSTC/XLX 1
LTC Petucci
Kirtland AFB, NM
87117

USA SDC/DASD-H-SB 1
Larry Tubbs
P. O. Box 1500
Huntsville, AL
35807

ANSER Corp 1
Suite 800
Crystal Gateway 3
1215 Jefferson Davis Highway
Arlington, VA 22202

AFOTEC/XPP 1
Capt Wrobel
Kirtland AFB, NM
87117

AF Space Command/XPXIS 1
Peterson AFB, CO
80914-5001

Director NSA 1
V43 George Hoover
9800 Savage Road
Ft. George G. Meade, MD
20755-6000